



UNIVERSIDADE FEDERAL DE PERNAMBUCO  
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO  
CENTRO DE INFORMÁTICA

---



UMA ANÁLISE DE TIPOS DE CONSULTAS E  
ESQUEMAS PARA REALIZAÇÃO DE SINTONIA  
EM BANCOS DE DADOS

---

TRABALHO DE GRADUAÇÃO

**Aluno:** Luis Ricardo Celestino de Souza (lracs@cin.ufpe.br)

**Orientadora:** Valéria Cesário Times (vct@cin.ufpe.br).

16 de agosto de 2005

## **Assinaturas**

Este Trabalho de Graduação é resultado dos esforços do aluno Luis Ricardo Celestino de Souza, sob a orientação da professora Valéria Cesário Times, na participação do desenvolvimento do trabalho “Uma Análise de Tipos de Consultas e Esquemas para Realização de Sintonia em Banco de Dados”, conduzido pelo Centro de Informática da Universidade Federal de Pernambuco. Todos abaixo estão de acordo com o conteúdo deste documento e os resultados deste Trabalho de Graduação.

---

**Luis Ricardo Celestino de Souza**

---

**Valéria Cesário Times**

# Agradecimentos

Primeiramente, gostaria de agradecer a meus pais, Salathiel Celestino de Souza e Inêz Bezerra Celestino de Souza. Estes dedicaram boa parte de suas vidas para que eu recebesse uma boa educação, e pudesse assim, atingir este nível de escolaridade.

Meus sinceros agradecimentos, especialmente, à minha orientadora e professora Valéria Cesário Times que sempre esteve presente, confiante, ajudando e foi bastante paciente nas inúmeras correções desta monografia.

À minha namorada, Luciana Pereira Oliveira, pelo amor, pela companhia e pela amizade.

Agradeço a meus irmãos, Petrus Clavio Celestino de Souza e Leonardo Cesar Celestino de Souza, por terem sido compreensivos e me apoiarem durante todos os momentos. Agradeço também a todos os familiares e amigos.

Um agradecimento especial também a Fábio Ávila, por gerar e disponibilizar a base de dados exemplo, utilizada para a simulação dos experimentos, cujos resultados estão exibidos nesta monografia.

Aos professores do Centro de Informática da UFPE pelo conteúdo ensinado ao longo destes 5 anos, principalmente aos da área de banco de dados.

Finalmente, agradeço a todos que contribuíram para o término deste trabalho de graduação, seja de forma direta ou indireta.

## Resumo

O aumento da complexidade dos sistemas requerendo cada vez mais uma maior quantidade de processamento levou à necessidade de otimização de performance das aplicações. Em grande parte destes sistemas, o desempenho da aplicação está relacionado com a performance do sistema de banco de dados. Isto ocorre, principalmente, na maneira em que são formuladas as consultas e como está estruturado o esquema da base de dados.

O esquema do banco de dados é crucial para o desempenho satisfatório das consultas. Existe uma forte relação entre ambos, de modo que a forma como o esquema deve ser feito depende das consultas que serão submetidas à base de dados. Desta forma, este trabalho tratará não só a análise de formas de consultas e esquemas separadamente, mas também como as consultas afetam os esquemas e vice-versa.

O trabalho apresentado nesta monografia, faz uma análise sobre um levantamento de um conjunto de princípios para a sintonia de esquemas e consultas. Alguns princípios também foram adicionados. E a partir da análise dos resultados obtidos em uma simulação, pôde-se ter um conjunto de princípios validados.

## **Abstract**

The existing increase of the computing system complexity requesting each turn a more quantity of processing has created a need for the performance optimization of many database applications. For most of these systems, the application performance is connected with the efficiency of database systems. This is mainly due to the way in which the user queries are formulated and the database schema is structured.

The database schema is critical for obtaining good performance results for the queries processing. There is a strong relationship between both, the database schema and the query processing, and as a result, the database schema design depends upon the queries submitted to the database system. Thus, this work aims to separately analyse the queries types and schema designs, and also, to investigate how the queries specifications may affect the database schemas and vice-versa.

The work presented in this monograph, makes an analysis on a survey of a set of principles for the tuning of database schemas and queries. Some principles had been also added. From the analysis of the results gotten in a simulation, it has been obtained a set of validated principles.

# Sumário

Índice de Figuras .....	9
Índice de Quadros .....	10
1. Introdução.....	13
1.1. Motivação .....	14
1.2. Objetivos.....	15
1.3. Estrutura .....	16
2. Sintonia de BD .....	17
2.1. Conceitos Básicos.....	20
2.1.1. Planos e Métodos de Acesso .....	21
2.1.2. Modelos de Dados .....	22
2.1.3. Auto-sintonia .....	23
2.2. Tipos de Sintonia .....	24
2.2.1. Otimização de Consultas .....	24
2.2.2. Otimização de Esquemas.....	26
2.3. Funcionalidades de SQL Importantes para a Sintonia.....	29
2.4. Trabalhos Correlatos .....	31
2.4.1. Comparação entre este trabalho e trabalhos correlatos .....	32
2.5. Conclusão .....	33
3. Princípios para Sintonia de Esquemas e Consultas .....	35
3.1. Esquema .....	35
3.1.1. Índices.....	36
3.1.2. Partições .....	40
3.1.3. Tipos dos Campos .....	41
3.1.4. Paralelismo .....	42
3.1.5. Normalização x Desnormalização .....	44
3.1.6. Cache de Tabelas .....	45
3.1.7. Clusters .....	46
3.1.8. Esquemas Distribuídos .....	46

3.2.	Consulta.....	47
3.2.1.	Operadores de Ordenação.....	48
3.2.2.	Junções .....	50
3.2.2.1.	<i>Junção NESTED LOOPS</i> .....	51
3.2.2.2.	<i>Junção SORT-MERGE</i> .....	51
3.2.2.3.	<i>Junção CLUSTER</i> .....	52
3.2.2.4.	<i>Junção HASH</i> .....	52
3.2.2.5.	<i>Junção de índices</i> .....	53
3.2.3.	Sub-consultas.....	53
3.2.4.	Hints .....	55
3.2.4.1.	<i>CHOOSE</i> .....	55
3.2.4.2.	<i>RULE</i> .....	56
3.2.4.3.	<i>FULL</i> .....	56
3.2.4.4.	<i>INDEX</i> .....	57
3.2.4.5.	<i>ORDERED</i> .....	57
3.2.4.6.	<i>DRIVING_SITE</i> .....	58
3.2.4.7.	<i>USE_NL</i> .....	58
3.2.4.8.	<i>PARALLEL</i> .....	59
3.2.4.9.	<i>NOPARALLEL</i> .....	59
3.2.4.10.	<i>CACHE</i> .....	59
3.2.4.11.	<i>NOCACHE</i> .....	60
3.2.5.	Suprimindo índices.....	60
3.2.6.	Cláusulas e restrições duplicadas .....	62
3.2.7.	Consultas Distribuídas.....	63
3.2.8.	WHERE x HAVING .....	63
3.3.	Conclusão .....	64
4.	Simulação e Análise .....	66
4.1.	Projeto do Experimento .....	66
4.2.	Resultados dos Experimentos.....	70
4.2.1.	Índices seletivos x índices não-seletivos .....	71
4.2.2.	Tipos de campos .....	73

4.2.3.	Atributo CACHE .....	74
4.2.4.	Eliminação de Ordenação .....	75
4.2.5.	Junções NESTED LOOPS x Junções SORT-MERGE .....	76
4.2.6.	Supressão de Índices.....	78
4.2.7.	Cláusulas e Restrições Duplicadas .....	80
4.2.8.	WHERE x HAVING .....	81
4.2.9.	Uso de clusters.....	82
4.2.10.	Normalização x Desnormalização .....	84
4.3.	Análise dos Resultados.....	86
4.3.1.	Índices seletivos x índices não-seletivos .....	86
4.3.2.	Tipos de campos .....	86
4.3.3.	Atributo CACHE .....	87
4.3.4.	Eliminação de Ordenação.....	87
4.3.5.	Junções NESTED LOOPS x Junções SORT-MERGE .....	87
4.3.6.	Supressão de Índices.....	88
4.3.7.	Cláusulas e Restrições Duplicadas .....	88
4.3.8.	WHERE x HAVING .....	89
4.3.9.	Uso de clusters.....	89
4.3.10.	Normalização x Desnormalização .....	89
4.4.	Conclusão .....	90
5.	Conclusão .....	92
5.1.	Principais Contribuições.....	92
5.2.	Trabalhos Futuros .....	92
	Referências .....	94

# Índice de Figuras

Figura 2.1 - Exemplo de plano de consulta. ....	22
Figura 2.2 - Exemplo de dependência funcional. ....	27
Figura 3.1 - Consulta executada paralelamente (modificada de [Niemec03]). ....	43
Figura 4.1 - Modelo conceitual da base de dados [TPC05]. ....	68

# Índice de Quadros

Quadro 2.1 - Criando um índice sobre uma tabela. ....	29
Quadro 2.2 – Consultando os índices existentes sobre uma tabela. ....	29
Quadro 2.3 – Consultando a quantidade de valores distintos em um campo indexado. ....	29
Quadro 2.4 – Especificando o grau de paralelismo na criação de uma tabela. ....	30
Quadro 2.5 – Criação de uma tabela com os registros disponíveis na CACHE.....	30
Quadro 2.6 – Criando um cluster no esquema físico.....	30
Quadro 2.7 – Criando duas tabelas dentro de um cluster. ....	31
Quadro 2.8 – Criando um cluster no esquema físico.....	31
Quadro 3.1 – Consulta utilizando função que altera campo.....	37
Quadro 3.2 – Consulta utilizando função que altera valor. ....	37
Quadro 3.3 – Criando um índice baseado em função.....	37
Quadro 3.4 – Consulta que utiliza índice baseado em função.....	37
Quadro 3.5 – Exibindo estatísticas sobre ordenação. ....	49
Quadro 3.7 – Exemplos de tipos de sub-consultas. ....	54
Quadro 3.8 – Usando o hint CHOOSE.....	56
Quadro 3.9 – Usando o hint RULE. ....	56
Quadro 3.10 – Usando o hint FULL.....	57
Quadro 3.11 – Usando o hint INDEX. ....	57
Quadro 3.12 – Usando o hint ORDERED.....	58
Quadro 3.13 – Usando o hint DRIVING_SITE. ....	58
Quadro 3.14 – Usando o hint USE_NL.....	58
Quadro 3.15 – Usando o hint PARALLEL. ....	59
Quadro 3.16 – Usando o hint NOPARALLEL. ....	59
Quadro 3.17 – Usando o hint CACHE. ....	60
Quadro 3.18 – Usando o hint NOCACHE. ....	60
Quadro 3.19 – Reescrita de consulta utilizando NOT EQUAL. ....	62
Quadro 3.20 – Utilizando restrições duplicadas.....	63
Quadro 4.1 – Comparação entre ferramentas para a simulação. ....	67

Quadro 4.2 – Utilizando o comando ANALIZE para atualizar estatísticas. ....	69
Quadro 4.3 – Consulta restringida pelo campo c_state. ....	71
Quadro 4.4 – Criando índice sobre o campo c_state da tabela customer. ....	71
Quadro 4.5 – Plano de Consulta do comando do quadro 4.3, sem índice. ....	71
Quadro 4.6 – Plano de Consulta do comando do quadro 4.3, com índice.....	72
Quadro 4.7 – Consulta restringida pelo campo c_credit. ....	72
Quadro 4.8 – Criando índice sobre o campo c_credit da tabela customer. ....	72
Quadro 4.9 – Plano de Consulta do comando do quadro 4.7, sem índice. ....	72
Quadro 4.10 – Plano de Consulta do comando do quadro 4.7, com índice.....	73
Quadro 4.11 – Consultando a tabela item. ....	73
Quadro 4.12 – Alterando o tipo de dado do campo i_name da tabela item.....	73
Quadro 4.13 – Plano de Consulta do comando do quadro 4.11, antes da alteração do campo i_name. ....	73
Quadro 4.14 – Plano de Consulta do comando do quadro 4.11, após a alteração do campo i_name. ....	74
Quadro 4.15 – Consultando a tabela orders.....	74
Quadro 4.16 – Alterando a tabela orders para ser armazenada em CACHE.....	74
Quadro 4.17 – Plano de Consulta do comando do quadro 4.15, sem uso do atributo CACHE.....	75
Quadro 4.18 – Plano de Consulta do comando do quadro 4.15, após primeira execução usando o atributo CACHE.....	75
Quadro 4.19 – Plano de Consulta do comando do quadro 4.15, após segunda execução usando o atributo CACHE.....	75
Quadro 4.20 – Consultando a tabela customer, utilizando as cláusulas GROUP BY e ORDER BY. ....	75
Quadro 4.21 – Consultando a tabela customer, utilizando a cláusulas GROUP BY. ....	76
Quadro 4.22 – Plano de Consulta do comando do quadro 5.20. ....	76
Quadro 4.23 – Plano de Consulta do comando do quadro 5.21. ....	76
Quadro 4.24 – Junção NESTED LOOPS entre district e customer. ....	77
Quadro 4.25 – Junção MERGE-SORT entre district e customer. ....	77
Quadro 4.26 – Plano de Consulta do comando do quadro 4.24. ....	77

Quadro 4.27 – Plano de Consulta do comando do quadro 4.25. ....	78
Quadro 4.28 – Consulta restringida por campo indexado. ....	78
Quadro 4.29 – Consulta restringida por campo indexado usando IS NULL. ....	78
Quadro 4.30 – Consulta restringida por campo indexado usando IS NOT NULL. ....	79
Quadro 4.31– Plano de Consulta do comando do quadro 4.28. ....	79
Quadro 4.32 – Plano de Consulta do comando do quadro 4.29. ....	79
Quadro 4.33 – Plano de Consulta do comando do quadro 4.30. ....	79
Quadro 4.34 – Consultando a chave primária da tabela orders usando DISTINCT. ....	80
Quadro 4.35 – Consultando a chave primária da tabela orders. ....	80
Quadro 4.36 – Plano de Consulta do comando do quadro 4.34. ....	80
Quadro 4.37 – Plano de Consulta do comando do quadro 4.35. ....	80
Quadro 4.38 – Utilizando a cláusula HAVING. ....	81
Quadro 4.39 – Utilizando a cláusula WHERE. ....	81
Quadro 4.40 – Plano de Consulta do comando do quadro 5.38. ....	81
Quadro 4.41 – Plano de Consulta do comando do quadro 5.39. ....	82
Quadro 4.42 – Criando o cluster customer_history. ....	82
Quadro 4.43 – Criando o cluster customer_history. ....	82
Quadro 4.44 – Consulta envolvendo uma junção entre customer e history. ....	83
Quadro 4.45 – Consulta envolvendo uma junção entre customer2 e history2. ....	83
Quadro 4.46 – Plano de Consulta do comando do quadro 5.44. ....	83
Quadro 4.47 – Plano de Consulta do comando do quadro 5.45. ....	83
Quadro 4.48 – Consultando dados das tabelas customer e history. ....	84
Quadro 4.49 – Consultando dados da tabela cust_hist. ....	84
Quadro 4.50 – Consultando a tabela customer. ....	84
Quadro 4.51 – Consultando dados existentes na customer através de cust_hist. ....	85
Quadro 4.52 – Plano de Consulta do comando do quadro 5.48. ....	85
Quadro 4.53 – Plano de Consulta do comando do quadro 5.49. ....	85
Quadro 4.54 – Plano de Consulta do comando do quadro 5.50. ....	85
Quadro 4.55 – Plano de Consulta do comando do quadro 5.51. ....	85
Quadro 4.56 – Resumo da simulação dos princípios. ....	91

# 1. Introdução

Sintonia (ou ajuste) em Banco de Dados (BD) [Sasha03] significa ajustar o sistema de banco de dados [Elmasri99, Korth01] de forma a obter uma melhor performance, garantindo um desempenho satisfatório para a aplicação em questão. Existem várias formas que podem ser empregadas para realização da sintonia, muitas dessas já disponibilizadas pelo próprio Sistema de Gerenciamento de Banco de Dados (SGBD). Essas formas variam entre si de acordo com o nível da arquitetura do sistema ao qual pertencem.

A otimização pode ser feita na parte de hardware, porém este tipo de otimização provavelmente tem um custo alto. Na parte de software, podem ser realizados quatro tipos de modificações: sistema operacional, SGBD, aplicação e banco de dados. Modificações do Sistema Operacional (SO) geralmente inclui a troca do SO ou a atualização (ou configuração) do mesmo de forma adequada. Modificações do SGBD podem incluir mudanças de configurações sobre o mesmo ou a troca de SGBD.

Dentre as ações para melhorar a performance, implementadas pelo SGBD, pode-se citar: escolha do melhor plano de execução de uma consulta de usuário e seleção de um mecanismo de bufferização de dados adequado. Neste caso, o próprio SGBD possuindo um otimizador para execução das consultas escolhe, dentre diferentes planos de consulta, um destes baseado em determinadas heurísticas e estatísticas sobre a base de dados. Cada plano de execução é formado por operações básicas e geram saídas equivalentes.

Por outro lado, outras formas de otimização devem ser implementadas pelo Administrador de Banco de Dados (ABD), tal como o aprimoramento do esquema ou pelo desenvolvedor da aplicação, tais como a criação e otimização de consultas. Formas de sintonia de esquemas, de consultas e de esquemas e consultas em conjunto constituem o foco da investigação a ser feita durante o desenvolvimento deste trabalho de graduação.

## 1.1. Motivação

O aumento da complexidade dos sistemas requerendo cada vez mais uma maior quantidade de processamento levou à necessidade de otimização de performance das aplicações. A partir daí, surgiram vários obstáculos e inúmeras tentativas para alcançar um desempenho desejado [Lehman86, Joshi01, Ganski87, Kim82, Oracle01].

Empresas desenvolvedoras de software deixam de efetuar algumas otimizações para cortar custos e consideram a realização de testes de baixa carga, como satisfatória, uma vez que dão uma importância ao número de funcionalidades implementadas. Porém, quando o sistema é submetido a uma carga maior de dados e processamento, observa-se que a performance almejada não é atendida. Estes sistemas podem tornar-se inviáveis dependendo do nível de desempenho alcançado e se técnicas de sintonia em BD não forem utilizadas para minimizar o problema.

Algumas empresas insatisfeitas com o tempo de resposta das aplicações, atualizam os equipamentos para obter um melhor desempenho dos sistemas. Porém, o custo do hardware é alto, especialmente comparado com o investimento necessário para realizar otimizações de software, para obter um melhoramento equivalente. Além disto, algumas melhorias de desempenho alcançadas por software podem não ser obtidas através de hardware.

É certo que a questão do ajuste oferece vários obstáculos à primeira vista. A otimização de desempenho dos sistemas com certeza aumenta os custos de desenvolvimento do software, pois requer mais tempo para que o produto seja desenvolvido e em alguns casos, exige mão de obra especializada. Portanto, é difícil desenvolver aplicações com a realização de sintonia sem afetar o custo geral de implementação do sistema. Porém, uma vez tendo feito isto, haverá ganhos de produtividade ao longo do uso do software e redução de custos na implementação do hardware.

Em grande parte destes sistemas, o desempenho da aplicação está relacionado com a performance do sistema de banco de dados. Isto ocorre principalmente, na maneira em que são formuladas as consultas e como está estruturado o esquema da base de dados. O esquema do banco de dados é crucial para o desempenho satisfatório das consultas. Existe

uma forte relação entre ambos, na qual a forma como o esquema deve ser feito depende das consultas que serão submetidas à base de dados. Qualquer nova consulta adicionada a um sistema existente poderá afetar o esquema, caso este não dê suporte à consulta, ou promova a sua execução de forma indesejada. Desta forma, este trabalho realizará não só uma análise de formas de consultas e esquemas separadamente, mas investigará também como as consultas afetam os esquemas e vice-versa. Este aspecto tem sido pouco abordado pela academia na área de sintonia em BD.

## 1.2. Objetivos

O trabalho proposto neste documento tem como objetivo estudar o impacto das diferentes formas de criação de esquemas e consultas no desempenho do sistema de banco de dados. Para isto, será feita uma pesquisa de trabalhos correlatos na área de sintonia de banco de dados em esquemas e consultas.

A idéia é que esta análise forneça subsídios para identificação de um conjunto de princípios que devam ser levados em consideração ao realizar ajustes em esquemas, em consultas ou em ambos. Na seqüência, para validação das idéias propostas neste trabalho de graduação, uma simulação sobre uma base de dados exemplo, utilizando o SGBD *Oracle 9i* [Loney01] também será feita. A escolha deste SGBD se deve ao fato dele possuir várias formas de otimização de consultas, ser bastante eficiente e ser amplamente usado no mercado [Silva03], estar disponível para realização dos testes e ser uma ferramenta de conhecimento prévio do aluno. Porém, é possível que muitos dos resultados obtidos possam ser generalizados para outros SGBD pelo modelo de dados relacional e por SQL terem se tornado um padrão.

Os resultados da simulação serão analisados a fim de extrair novas informações e ilustrar ou validar os princípios propostos. Será observado o impacto da criação de consultas sobre o esquema do banco de dados, visto que ambos estão diretamente relacionados, bem como uma análise de desempenho das variações dos esquemas e das consultas em separado será também realizada.

### 1.3. Estrutura

Nesta seção, será apresentada a estrutura deste trabalho. No capítulo 2, são abordados algumas definições e conceitos relacionados com a área de sintonia de BD, bem como os tipos de sintonia a serem tratados aqui, algumas funcionalidades de SQL importantes para a sintonia e trabalhos correlatos a esta monografia. Os resultados do levantamento de um conjunto de princípios de sintonia de esquemas e consultas são mostrados no capítulo 3. No capítulo 4, exibimos resultados da simulação realizada durante o desenvolvimento deste trabalho, juntamente com uma análise feita sobre alguns princípios exibidos no capítulo anterior. Este quarto capítulo também inclui as decisões de projeto do experimento realizado, incluindo a escolha de uma ferramenta e de uma base de dados para realização do mesmo. Finalmente, no capítulo 5 apresentamos uma conclusão sobre o trabalho efetuado.

## 2. Sintonia de BD

Existem várias formas de melhorar a performance de um sistema de Banco de Dados (BD). Algumas destas formas não estão diretamente relacionadas com a aplicação ou com o Sistema de Gerenciamento de Banco de Dados (SGBD), englobando assim, várias áreas multidisciplinares. Dentre essas áreas estão o hardware, o sistema operacional (SO), a gerência de memória e o acesso a discos [Lifschitz04].

O objetivo da sintonia é simples: atingir um melhor desempenho através da especificação e execução de ajustes. Porém, a sua realização tende a ser complexa devido a várias dificuldades. Por exemplo, nem sempre é fácil entender o problema e descobrir sua verdadeira origem, e para isto, são necessários uma análise detalhada sobre o mesmo e um investimento de tempo no seu diagnóstico.

A sintonia (*tuning*) do banco de dados pode ser feita adicionando-se hardware ao sistema, tais como [Sasha03]:

- Aumentando o tamanho da memória – consiste no aumento do espaço disponível para o *buffer pool* que é uma área de memória alocada para gerenciamento do banco de dados, como cache de tabelas, índices e páginas. O *buffer pool* provê a otimização do sistema de banco de dados, pois os dados mantidos nele são acessados de forma mais rápida e os acessos à memória secundária são minimizados. Assim, o gerenciador do banco de dados decide, através de uma política de gerenciamento de *buffer*, quando e quais dados devem ser substituídos na memória.
- Acrescentando novos discos – quanto maior for o espaço disponível de armazenamento secundário, menor será o número de realocações de espaços, diminuindo-se a fragmentação dos dados em diversos segmentos não adjacentes. Além disso, pode-se utilizar uma Matriz Reduntante de Discos Independentes (*Redundant Arrays Of Independent Disks – RAID*) [Sasha03, Garcia00] que geralmente proporciona um melhor desempenho para as aplicações envolvidas.
- Adicionando novos processadores – um sistema multiprocessado é o modo mais barato de conectar vários sistemas computacionais diferentes [Sasha03] e o

processamento de cada transação executada concorrentemente pode ser feito em um processador diferente, diminuindo assim, a disputa por um processador e gerando um melhor desempenho.

Os SGBD também tentam auxiliar neste aspecto de sintonia, através do uso de heurísticas, da criação de estruturas de dados para realização da otimização, do gerenciamento de memória disponível para o sistema de banco de dados e da escolha entre diferentes planos de execução de uma consulta (chamados simplesmente de planos de consulta). Porém, para que o SGBD possa desempenhar este papel com sucesso, ele precisa estar bem configurado.

Em uma aplicação de sistema de banco de dados, o SGBD tem grande influência no seu desempenho, pois ele abstrai os usuários de muitas questões referentes ao gerenciamento da performance. Entre as atividades executadas por ele, está o controle de concorrência, o qual gerencia a granularidade dos bloqueios e realiza a eliminação de bloqueios desnecessários para que seja evitado o prolongamento da espera por alguma transação que esteja concorrendo por um mesmo recurso [Sasha03].

Para que seja atingido um desempenho satisfatório, o refinamento da performance deve ser realizado desde o início da produção do sistema de aplicação, incluindo as etapas de análise, desenvolvimento e implementação. O desempenho está diretamente relacionado com a qualidade do sistema, visto que esta característica é geralmente um dos requisitos não funcionais do mesmo.

Um outro fator que afeta o desempenho do sistema de banco de dados, e por isso deve ser levado em consideração, é a configuração do SO. Parâmetros, como o *time-slice* (tempo de execução de um processo executado concorrentemente com outros processos em um ambiente multi-tarefas) e o tamanho da memória RAM disponível para o SGBD, devem ser observados na configuração do SO.

Em [Fernandes02], um conjunto de tarefas que devem ser realizadas durante cada etapa do ciclo de desenvolvimento de software é apresentado:

- Na definição e implementação das regras de negócio – a escolha das regras de negócio, a partir de um conhecimento preciso sobre a aplicação, deve considerar os

primeiros pontos focais passíveis de implementação, na busca por otimização de aplicações;

- Na etapa de modelagem conceitual dos dados – nesta fase, estamos determinando as entidades que o futuro banco de dados possuirá. Deve-se, a partir do modelo conceitual, verificar a possibilidade do uso de técnicas de normalização e avaliar a necessidade de desnormalizações;
- Na etapa de avaliação do projeto da aplicação – Nesta fase, define-se como a aplicação irá atender as regras de negócio estabelecidas na primeira etapa. Esta fase é de grande importância, pois nela, determina-se quais as funcionalidades que farão acessos ao banco de dados, quais as expectativas de tempo de resposta e quais as restrições de tempo de resposta do sistema;
- Durante o refinamento do modelo de dados físico – Pode-se determinar quais índices devem ser incluídos no modelo, para que as funcionalidades com restrições de desempenho possuam um acesso aos dados eficiente, sem que haja prejuízo significativo nas atualizações dos dados. Os tipos de índices também são escolhidos nesta etapa de acordo com os tipos das consultas a serem realizadas;
- Durante a análise da utilização de SQL nas aplicações – Esta etapa, também conhecida como reescrita de consultas, consiste em um estudo de performance sobre as diferentes formas de escritas de consultas que geram um resultado equivalente. Pequenas mudanças nos comandos desta linguagem podem trazer grandes diferenças no desempenho de um programa de aplicação;
- Em outras etapas que também influenciam o resultado final – Na avaliação das necessidades de hardware, na qual deve-se observar a escalabilidade do sistema para determinar a demanda de hardware a médio e longo prazos, evitando a necessidade de grandes mudanças devido a futuras expansões. Também, na avaliação do banco de dados feita pelo Administrador de Banco de Dados (ABD), o qual definirá a frequência de avaliação e a necessidade de manutenção do sistema de banco de dados.

Neste capítulo, os conceitos básicos utilizados neste trabalho serão descritos juntamente com uma breve discussão sobre as principais áreas focadas neste estudo. A

seção 2.1 aborda os conceitos básicos utilizados na área de sintonia de banco de dados. A seção 2.2 explica as formas de otimização de consultas e porquê utilizá-las, e também, descreve possíveis mudanças nos esquemas e como essas modificações podem afetar o desempenho. A seção 2.3 explica algumas funcionalidade de SQL importantes para a sintonia de BD. Alguns trabalhos correlatos e seus benefícios são citados na seção 2.4. Por último, a seção 2.5 finaliza este capítulo com uma breve conclusão sobre o assunto abordado.

## 2.1. Conceitos Básicos

SQL é uma linguagem para interface com bancos de dados relacionais [Elmasri99], isto é, todos os usuários e programas que desejarem realizar alguma tarefa no banco de dados devem fornecer comandos escritos nesta linguagem. Ela se tornou um padrão, para bancos de dados relacionais, publicado pela *American National Standard Institute* (ANSI) e pela *International Standards Organizaton* (ISO). A SQL do SGBD *Oracle 9i* usado nesta pesquisa é uma extensão desta especificação padrão.

A linguagem SQL é dividida em três partes: A Linguagem de Manipulação de Dados (*Data Manipulation Language - DML*), a Linguagem de Definição de Dados (*Data Definition Language - DDL*) e a Linguagem de Controle de Dados (*Data Control Language - DCL*). O foco deste trabalho consiste nas linguagens DML (para a sintonia de consultas) e DDL (para a sintonia de esquemas).

A realização de sintonia em BD é uma atividade contínua, que faz parte da manutenção do sistema, que se estende ao longo do ciclo de vida da base de dados, e é realizada enquanto o banco de dados e as aplicações envolvidas se mantiverem em evolução e quando problemas relativos ao desempenho forem detectados. Em geral, à medida que os requisitos funcionais e não-funcionais do sistema de banco de dados se alteram, torna-se necessário acrescentar, retirar tabelas existentes e reorganizar alguns arquivos, alternando os métodos de acesso primário ou eliminando antigos índices e construindo novos. Algumas consultas e transações podem precisar ser reescritas para

melhorar o desempenho. A sintonia do banco de dados é feita continuamente, enquanto o banco de dados existir, quando os problemas de desempenho forem descobertos e enquanto os requisitos continuarem a se modificar [Elmasri99]. Por exemplo, uma consulta que esteja sendo realizada durante o intervalo de tempo de 4 segundos pode ter seu desempenho considerado como insatisfatório após a mudança de requisito do sistema de término da mesma para 3 segundos. Outro exemplo seria uma restrição para o sistema de que a mesma consulta tivesse que durar no máximo 5 segundos, e após um certo período de tempo a consulta passasse a durar 6 segundos, por causa do aumento do número de registros pesquisados na base.

### 2.1.1. Planos e Métodos de Acesso

O plano de consulta descreve como uma consulta é executada, e é formado por uma árvore contendo as operações básicas (junção, projeção, seleção, ordenamento, dentre outras) que serão usadas no seu processamento. Para cada operação, existe uma indicação de como ela será executada (incluindo *index nested loop*, *sort*, *index* e *simple scan* [Manber89]). A Figura 2.1 mostra uma consulta e um exemplo de um plano de consulta para a mesma.

Os métodos de acesso são operações de retorno otimizado de dados utilizados na execução de um comando SQL. Os principais tipos de métodos de acesso são:

- Varredura sequencial (*full table scan*) – no qual uma tabela específica é totalmente percorrida para retornar os valores de uma busca;
- Busca baseada em *rowid* - tendo como entrada um identificador de registro (*rowid*), pode-se obter de forma direta um registro de uma tabela contida na base de dados;
- Varredura indexada única - utiliza um índice existente no banco de dados para retornar tuplas de uma tabela restringida pelo índice utilizado. Neste tipo de varredura, cada valor indexado possui uma entrada no índice; e
- Varredura indexada por faixa de valores - semelhante à varredura indexada única,

porém, cada entrada no índice está associada a uma faixa de valores na tabela indexada.

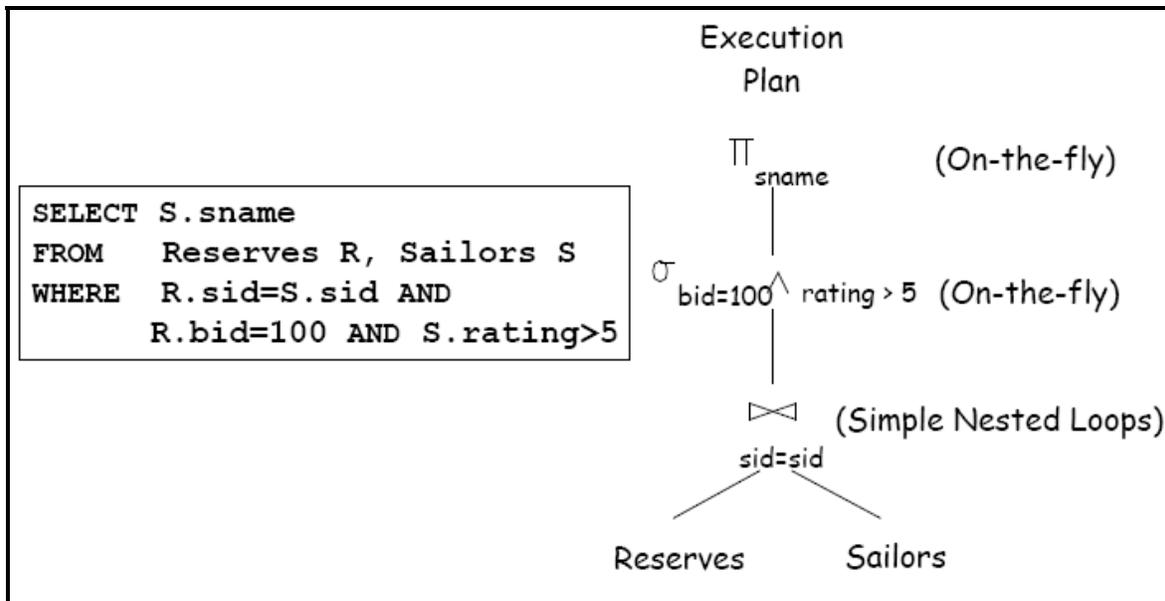


Figura 2.1 - Exemplo de um plano de consulta [Kemme05].

### 2.1.2. Modelos de Dados

O modelo de dados é um conjunto de conceitos utilizados para definir formalmente uma representação do mundo real através de uma estrutura de dados. Ele auxilia no entendimento dos dados e dos relacionamentos entre as entidades que compõem a base de dados. Além disso, serve como um modelo formal para comunicação entre os desenvolvedores e ABD do sistema.

Um modelo de dados deve possuir as seguintes características de acordo com Elmasri [Elmasri99]:

- Expressividade – deve ser expressivo ao ponto de se poder distinguir entre os relacionamentos, os tipos de dados e as restrições;
- Simplicidade e legibilidade – deve ser simples e de fácil compreensão para usuários

que sejam especialistas ou não;

- Minimalidade – deve possuir uma quantidade mínima de entidades e atributos para que sejam completos e sem sobreposição;
- Representação diagramática – deve ter um esquema conceitual gráfico de fácil interpretação; e
- Formalidade – deve ter sido representado através de uma especificação formal, de modo que seus conceitos sejam não ambíguos.

Estes modelos são geralmente classificados em três tipos, de acordo com o nível de abstração [Elmasri99]:

- Modelo Conceitual – representa os dados em um formato de alto nível, sem se preocupar com restrições de implementação. É uma visão global das entidades e relacionamentos, sendo totalmente independente do tipo de SGBD escolhido para o desenvolvimento da aplicação de BD;
- Modelo Lógico – utilizado para obter um nível de abstração mais próximo da implementação que o modelo conceitual. Porém, este não é uma representação direta da implementação. A partir deste modelo, já se define o tipo de SGBD (Relacional, Hierárquico, Objeto-Relacional) a ser utilizado; e
- Modelo Físico – descreve como os dados estão armazenados no SGBD. É específico e totalmente dependente de cada SGBD e, nele, estão descritas as estruturas de armazenamento utilizadas, como sequencial, índices, *B-Trees* e *Hash Tables*.

### **2.1.3. Auto-sintonia**

A realização de adaptações automáticas sobre parâmetros, configurações e estruturas de um SGBD, de forma que haja um melhoramento da performance das consultas e atualizações sobre o mesmo, é chamada de *auto-sintonia de BD* [Lifschitz04]. A alocação

dos dados em memória principal, a coleta automática de estatísticas, o controle de múltiplas transações e mudanças sobre o esquema físico, visando um melhor desempenho, são alguns exemplos de auto-sintonia.

O módulo de otimização de consultas de um SGBD pode ser considerado como um componente de auto-sintonia. Isto porque ele pode armazenar estatísticas sobre os dados presentes nas bases de dados, e a partir destas, escolher a melhor opção entre as diversas alternativas existentes para a realização de uma consulta. O *Oracle 9i* também realiza ajustes para incrementar o desempenho das consultas executadas sobre os bancos de dados. Um destes ajustes é a criação de índices implicitamente, visando uma maior eficiência na execução de consultas, quando uma tabela é criada.

## 2.2. Tipos de Sintonia

Nesta parte, serão abordados dois tipos de sintonia de banco de dados. Na seção 2.2.1, é visto como geralmente é realizada a otimização de consultas. A seção 2.2.2 descreve a importância da otimização de esquemas. Assim como, ela é normalmente efetuada.

### 2.2.1. Otimização de Consultas

Apesar do próprio SGBD escolher o plano de acesso para cada consulta, precisamos definir se uma consulta precisa ser reescrita e de que forma ela deveria ser reescrita, pois algumas otimizações tornam-se bastante complexas até mesmo para o SGBD que pode não estar preparado para otimizar ao máximo, cada consulta. Além disso, o custo de se tentar otimizar uma consulta pode se tornar mais caro do que a utilização de plano de execução não otimizado.

Após mapear a consulta em rotinas básicas que serão aplicadas sobre o banco de

dados, são feitas várias combinações destas unidades em ordens de execução diferentes. Cada combinação é considerada um plano de execução distinto, e estas combinações possuem diferentes tempos de execução. Cabe ao SGBD escolher o plano de acesso mais eficiente, preferencialmente de forma que o tempo gasto na escolha seja menor que o ganho obtido na otimização.

A ordem da execução dos operadores básicos influencia no custo da consulta. Se possível, deve-se iniciar o processamento da consulta com os operadores de seleções e de projeções que restringem o conteúdo a ser utilizado em outras rotinas, e deve-se procurar adiar ao máximo, a utilização de alguns operadores binários como junções, uniões e produtos cartesianos que geram maiores quantidade de dados como resultado.

Os SGBD coletam estatísticas sobre os dados para tomar decisões sobre qual plano de execução deverá ser escolhido. Os tamanhos das tabelas, a quantidade de valores distintos por campo, e a frequência de uma determinada consulta são exemplos de estatísticas que auxiliam na criação de informações relevantes ao processo de sintonia.

Nenhum SGBD é perfeito o suficiente ao ponto de substituir um ABD, pois o primeiro é suscetível a falhas e não possui um conhecimento completo das aplicações que o utilizam. Várias aplicações podem utilizar a mesma base de dados e um melhoramento de performance para uma aplicação pode gerar um mal desempenho para outra aplicação. Sendo assim, cabe ao ABD decidir qual opção é a melhor escolha global.

Apesar de algumas otimizações serem feitas pelo SGBD, é possível que parte destas otimizações possam ser obtidas através da reescrita das consultas. Desta forma, é fundamental que as consultas sejam reescritas para que o trabalho e o tempo gasto do SGBD durante a otimização seja menor.

As consultas devem ser otimizadas desde o início do desenvolvimento da aplicação, dado que as alterações nesta fase possuem um custo menor do que as das fases posteriores. As alterações nas consultas podem requerer alterações no esquema da base de dados e isto pode ter um forte impacto se for feito após a fase do projeto do banco de dados, principalmente se várias aplicações fizerem uso da mesma base de dados. É preciso também antecipar o início dos testes destas consultas para que futuras modificações sejam evitadas ou minimizadas.

O desempenho de uma consulta pode variar bastante se existir ou não índices para

serem utilizados pela consulta. Porém, o uso de índices gera um *overhead* nas atualizações por causa da necessidade de atualização na tabela de índice. Deste modo, o ABD deve estimar o custo/benefício da adição de cada índice no modelo físico de dados.

### **2.2.2. Otimização de Esquemas**

O projeto da base dados é uma das fases do desenvolvimento de uma aplicação, na qual são criados os modelos de dados que informam como o dado a ser armazenado estará estruturado. É importante que o modelo de dados desenvolvido seja bastante estável, para que não ocorram mudanças no esquema físico durante a utilização do sistema.

Tais mudanças no esquema podem requerer modificações nas aplicações que utilizam a base de dados. Algumas restrições, como a de um sistema ter que estar quase sempre disponível, também podem dificultar a modificação do esquema. A criação de um modelo de dados o mais estável possível, economizará tempo de manutenção, durante o aumento da quantidade de dados contida na base.

A seguir, iremos descrever algumas escolhas e modificações no esquema que podem influenciar no desempenho de um sistema de banco de dados. Uma destas modificações é a escolha adequada do tipo de dados para um determinado campo. Nesta escolha, deve-se observar se os valores a serem cadastrados no campo utilizarão todo o espaço reservado para ele, pois pode ser que o espaço utilizado nesta coluna possa ser melhor aproveitado utilizando um outro tipo de dados. Além de melhorar a performance por minimizar o número total de bytes lidos pelo SGBD, pode-se economizar muito espaço se a quantidade de registros na tabela for razoavelmente grande.

A eliminação de campos desnecessários e a escolha da granularidade dos dados a serem armazenados também podem economizar espaço em disco pelo mesmo motivo anterior, além de melhorar a performance do sistema. Deve-se identificar a princípio, qual o nível de detalhamento que é relevante, para que dados desnecessários não sejam cadastrados na base.

Apesar da base de dados não estar populada na fase de projeto do banco de dados, o

ABD deve prever futuras estatísticas sobre ela e observar como serão as futuras consultas. Algumas consultas são executadas com frequência (consultas padrão) e geralmente, acessam uma determinada faixa de valor de um campo. Neste caso, pode-se atingir um melhor desempenho através do particionamento da tabela, pois desta forma não seria necessário consultar toda tabela para selecionar a faixa de valores desejada. Porém, o particionamento aumenta a complexidade do esquema e outras soluções, como a criação de índices, podem ser mais adequadas.

Considere os conjuntos *CPF* e *Nome* da Figura 2.2. Pode-se observar que existe uma dependência entre os valores destes conjuntos que pode ser expressa pela função  $f(CPF)=Nome$ . Ou seja, *Nome* é função do *CPF*, e esta dependência funcional é representada no Modelo Relacional por  $CPF \twoheadrightarrow Nome$ .

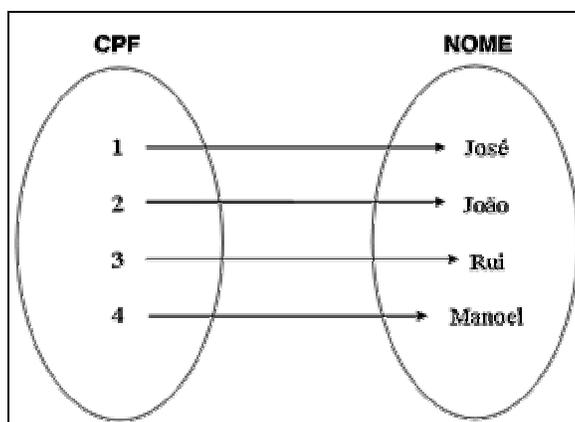


Figura 2.2 - Exemplo de dependência funcional.

Uma relação  $R$  está normalizada se toda dependência funcional  $X \twoheadrightarrow A$  envolvendo os atributos de  $R$  tem a propriedade de que  $X$  é uma chave de  $R$  [Sasha03]. Uma relação desnormalizada pode apresentar vários problemas, como redundância desnecessária de dados, perdas acidentais de informação, dificuldade de representação dos fatos, dependências transitivas entre atributos não chaves e inconsistência.

Apesar da desnormalização gerar vários problemas, pode-se obter com ela, um ganho de desempenho para a execução de consultas. Isto ocorre porque a quantidade de acessos ao disco é reduzida, por causa dos dados estarem em uma mesma tabela, além de

eliminar a necessidade de utilização do operador de junção para consultas em que anteriormente era necessária a junção das relações normalizadas.

Índices são estruturas auxiliares que visam permitir o acesso mais rápido aos dados [Lifschitz04], e eles estão associados a um ou mais campos de uma determinada tabela. Os índices são transparentes às consultas das aplicações, pois o resultado retornado pelo SGBD não varia com existência ou não de um índice. Porém, os desenvolvedores de aplicação devem possuir o conhecimento da ocorrência dos índices existentes no sistema de banco de dados, para que possam reescrever os comandos em SQL de forma otimizada.

Existem vários tipos de índices que geralmente variam de acordo com o SGBD e devem ser usados de acordo com o tipo da consulta que se deseja executar. A maioria desses índices podem ser classificados em duas categorias: cluster e não-cluster.

Os índices do tipo cluster em um atributo (ou conjunto de atributos) são índices em que a árvore de índice e os próprios dados formam uma única estrutura. A principal característica deste índice é que as páginas de dados da tabela são ordenadas fisicamente no banco de acordo com a chave do índice. Desta forma, o índice clusterizado permite uma boa performance em consultas que fazem uma busca em intervalos ou ordenação alfabética. Como ele ordena fisicamente as páginas, deverá haver um único índice cluster por tabela.

Os índices não-cluster possuem uma estrutura separada das páginas de dados das tabelas e liga-se a elas por ponteiros ou pela chave do índice clusterizado, se este existir na tabela. Como não ordena fisicamente os dados, normalmente sua criação é mais rápida. No caso de buscas pontuais, eles constituem uma boa escolha.

A criação de índices tende a otimizar as consultas realizadas sobre uma tabela, mas gera também um *overhead* para atualizações por causa da necessidade de modificar o índice. Por isso, deve-se verificar o impacto na criação de índices no sistema. Estatísticas sobre as frequências de consultas sobre um dado conjunto de atributos e de atualizações nesta tabela podem ser decisivas para tomada de decisões sobre a criação ou não de um índice.

## 2.3. Funcionalidades de SQL Importantes para a Sintonia

As principais funcionalidades de SQL que são importantes para realização de atividades de sintonia de BD são descritas nesta seção. Para isto, a sintaxe da linguagem disponível no SGBD *Oracle 9i* é utilizada. Uma das funcionalidades de SQL é a criação de índices sobre colunas em uma tabela, podendo melhorar o desempenho de algumas consultas. Para criar um índice sobre o campo *nome* de uma tabela *Funcionario*, utiliza-se o comando SQL mostrado no quadro 2.1.

**Quadro 2.1 - Criando um índice sobre uma tabela.**

```
CREATE INDEX func_ind1 ON Funcionario (nome);
```

Os usuários do *Oracle 9i* podem desejar obter uma lista dos índices existentes em uma tabela. No quadro 2.2, é apresentado como isto pode ser realizado para a tabela *Funcionario*.

**Quadro 2.2 – Consultando os índices existentes sobre uma tabela.**

```
SELECT table_name, index_name  
FROM user_indexes  
WHERE table_name='Funcionario';
```

Pode-se desejar saber a seletividade de um índice a partir da quantidade de valores distintos e existentes em uma coluna indexada de uma tabela. Para isto, pode-se consultar o valor da coluna *distinct\_keys* usando a visão *USER\_INDEXES* existente no SGBD *Oracle 9i*, de acordo com o que é mostrado no quadro 2.3.

**Quadro 2.3 – Consultando a quantidade de valores distintos em um campo indexado.**

```
SELECT index_name, distinct_keys  
FROM USER_INDEXES;
```

O *Oracle 9i* também permite que ao se criar uma tabela, seja definido o grau de paralelismo da mesma. No quadro 2.4, é mostrado como pode ser especificado o grau de

paralelismo de uma tabela no momento de sua criação.

**Quadro 2.4 – Especificando o grau de paralelismo na criação de uma tabela.**

```
CREATE TABLE Funcionario (  
    cpf NUMBER(8) NOT NULL,  
    nome VARCHAR2 (50),  
    data_nascimento DATE  
)  
PARALLEL 3;
```

Pode-se dar prioridade a uma tabela para a mesma ser armazenada em *CACHE*. O quadro 2.5 mostra como indicar ao SGBD, na criação de uma tabela, que esta deve ser armazenada em *CACHE*.

**Quadro 2.5 – Criação de uma tabela com os registros disponíveis na *CACHE*.**

```
CREATE TABLE Funcionario CACHE (  
    cpf NUMBER(8) NOT NULL,  
    nome VARCHAR2 (50),  
    data_nascimento DATE  
);
```

*Clusters* podem ser criados no *Oracle 9i* para que tabelas possam compartilhar os mesmos blocos de dados. No quadro 2.6, é criado um cluster no esquema físico para armazenar as tabelas *Funcionario* e *Departamento*. Então, no quadro 2.7, é mostrado como pode ser feita a criação de tais tabelas dentro do *cluster*. Finalmente, uma forma de criação de um índice sobre o *cluster* é exibida no quadro 2.8.

**Quadro 2.6 – Criando um cluster no esquema físico.**

```
CREATE CLUSTER cluster1 (id_departamento NUMBER(4))  
SIZE 600  
STORAGE (INITIAL 200K  
        NEXT 300K);
```

**Quadro 2.7 – Criando duas tabelas dentro de um cluster.**

```
CREATE TABLE Funcionario (  
    cpf NUMBER(8) PRIMARY KEY,  
    nome VARCHAR2 (50),  
    data_nascimento DATE,  
    id_departamento NUMBER(4)  
)  
CLUSTER cluster1;  
  
CREATE TABLE Departamento (  
    id_departamento NUMBER(4) PRIMARY KEY,  
    nome VARCHAR2 (20)  
)  
CLUSTER cluster1;
```

**Quadro 2.8 – Criando índice sobre um cluster no esquema físico.**

```
CREATE INDEX indicel  
ON CLUSTER cluster1;
```

## 2.4. Trabalhos Correlatos

Nesta seção são abordados alguns dos principais trabalhos relacionados a esta monografia. Estes trabalhos foram estudados, analisando as sugestões nas áreas de esquemas e consultas. Desta forma, podemos comparar os benefícios gerados por estes na área de sintonia de BD.

Em [Silva03] não é focada a parte de esquemas, pois o mesmo está voltado para a análise de consultas. Um assunto abordado é o uso de índices. O principal objetivo em [Silva03] é (1) investigar as diferentes formas de organização de consultas de modo que o desempenho de execução destas seja otimizado; e (2) analisar a influência do uso de índices sobre a velocidade de acesso a informações de algumas consultas.

Também, [Silva03] especifica e analisa diferentes formas de otimização de consultas. Além disso, a importância da necessidade da sintonia de BD e as principais dificuldades são bastante enfatizadas. O foco é voltado para a análise de formas de otimização de consultas. Por isso, é dado, uma maior ênfase nesta parte, do que na análise

de modificações no esquema. Alguns temas abordados são o uso de *hints*, junções, subconsultas, ordenações e supressão de índices.

Em [Lifschitz04], a sintonia de banco de dados é tratada de forma genérica. Assuntos como o uso de índices, particionamento e *clusters* são citados nesse trabalho. Porém, boa parte destes assuntos é vista superficialmente, já que não faz parte do escopo do mesmo. Procurou-se assim, em nossa pesquisa, detalhar e analisar os assuntos tratados em [Lifschitz04], além de abordar outros assuntos como o uso de tipos de campos diferentes, a execução paralela de consultas, as normalizações, as desnormalizações, o uso de tabelas em *CACHE* e a distribuição de dados. O objetivo fundamental do trabalho apresentado em [Lifschitz04] é exibir as principais técnicas de sintonia de BD e algumas ferramentas de auto-sintonia.

#### **2.4.1. Comparação entre este trabalho e trabalhos correlatos**

O trabalho mostrado em [Silva03] é voltado para o *Oracle 9i*. Nele são exibidos alguns tipos de junções utilizados por este SGBD. Detalhamos outros tipos de junções que não foram mencionadas nesse, como por exemplo junções *cluster* e junções de índices. Além disto, explicamos a questão de cláusulas e restrições duplicadas. Também, procuramos dar uma maior ênfase sobre o uso de *hints*, procurando mostrar outras formas de influenciar nos planos de consulta escolhidos pelo SGBD.

Em [Lifschitz04], a sintonia em banco de dados é apresentada de forma geral, abordando diferentes formas de ajustes em BD. Um dos assuntos destacados é a auto-sintonia de BD, que é apresentado como um modo de atribuir uma maior autonomia ao SGBD. São citados alguns exemplos de reescrita de consultas que também procuramos detalhar e analisar em nosso estudo. Este trabalho não está relacionado a um SGBD específico, por não ser de interesse do mesmo.

O trabalho apresentado nesta monografia faz um levantamento dos principais princípios apresentados por vários trabalhos correlatos [Lifschitz04, Silva03, Joshi01, Ganski87, Ozsu99], além de adicionar novos princípios no campo de sintonia de esquemas

e consultas. No nosso trabalho, procurou-se explicar outras formas de sintonia de esquemas. Analisando assim, outras interações existentes entre consultas e esquemas para melhorar o desempenho. Foram simulados também alguns destes princípios, de forma a obter um conjunto de princípios validados.

## 2.5. Conclusão

A sintonia do banco de dados abrange diversas áreas e por isso, torna-se uma tarefa bastante complexa, na qual o desenvolvedor da aplicação e o ABD devem possuir conhecimentos abrangentes para que as deficiências de performance possam ser resolvidas. Sendo assim, torna-se difícil descobrir a origem dos problemas de performance no momento em que eles surgem.

Para evitar que o sistema de banco de dados fique indisponível por muito tempo, a sintonia deve ser antecipada e não apenas ser feita após a implementação do sistema. O desempenho do sistema quando o mesmo for submetido a uma grande carga de conteúdo, deve também ser levado em consideração.

Apesar do incremento de hardware ajudar na sintonia do banco de dados, o custo da manutenção do hardware é caro e pode ser diminuído observando outros fatores que influenciam no desempenho dos sistemas, tais como o ajuste de esquemas e consultas.

O SGBD resolve muitas questões de sintonia, como o gerenciamento de memória, a otimização de consultas e o controle de concorrência, mas ele não é perfeito ao ponto de resolver todas as questões de desempenho de sistemas de BD existentes. A otimização de consultas consome tempo que deve ser usado da melhor forma possível, e para isto, as consultas devem ser reescritas de forma que a otimização feita pelo SGBD seja mais simples de ser executada.

O esquema da base de dados precisa ser desenvolvido corretamente antes do sistema ser colocado em execução, de forma que o mesmo seja o mais estável possível. Alterações futuras no esquema podem deixar sistemas indisponíveis e causar grandes prejuízos às empresas que os utilizam.

Dados estatísticos sobre o banco de dados são bastante úteis na tomada de decisões sobre esquemas e consultas. Eles são essenciais para saber dentre as diversas possibilidades existentes, qual a melhor opção para a sintonia do banco de dados.

Este trabalho foca o ajuste de esquemas e a interação deste com a sintonia de consultas. Através de uma análise dos princípios identificados em nosso estudo e dos resultados de uma simulação efetuada, durante o desenvolvimento deste trabalho, pôde-se obter alguns resultados interessantes que serão apresentados nos capítulos seguintes.

Neste capítulo, foram discutidos os principais conceitos de sintonia de BD, bem como algumas funcionalidades SQL importantes para a sintonia de BD e alguns trabalhos correlatos foram discutidos. No próximo capítulo, serão abordados e analisados diversos princípios para a sintonia de esquemas e consultas.

## 3. Princípios para Sintonia de Esquemas e Consultas

Nos capítulos anteriores, podemos observar que para a otimização do sistema de banco de dados deve-se considerar vários aspectos. Este capítulo aborda como é realizada a sintonia em banco de dados. Para isto, foram pesquisadas diversas referências na área de sintonia de esquemas e consultas [Niemec03, Lifschitz04, Joshi01, Ganski87, Ozu99]. Em seguida, foi feita uma análise sobre os princípios existentes nesta área, a fim de investigar grande parte destes princípios.

Como alguns aspectos são específicos de cada ferramenta, decidimos utilizar o *Oracle 9i* por ele ser um dos SGBD mais utilizados no mundo [Loney02], e por também estar disponível para a realização de nossos testes. Mesmo assim, acreditamos que muitos dos princípios discutidos neste capítulo, são aplicáveis a outros SGBD, dada a padronização de SQL.

O restante deste capítulo está organizado da seguinte forma. Na seção 3.1, são citadas modificações na parte do esquema que podem ser feitas para atingir uma melhor performance. Na seção 3.2, algumas formas possíveis de reescrita de consultas são analisadas para que o tempo de resposta da mesma seja minimizado. Finalmente, na seção 3.3, uma breve conclusão sobre o capítulo é apresentada.

### 3.1. Esquema

A criação de um esquema conceitual estável é uma das tarefas mais importantes para a sintonia de banco de dados. Escolhas erradas e feitas inicialmente, podem acarretar em grandes complicações no futuro, sendo difíceis de serem resolvidas. O esquema também deve ser escalável, a ponto de suportar futuras modificações sem grandes prejuízos.

A sintonia do esquema lógico depende das estatísticas sobre os dados a serem armazenados. Além disso, o tempo de resposta necessário depende de aplicação para aplicação. Algumas consultas também podem requerer menor tempo de resposta do que

outras. O ideal é que haja um único esquema, que estruture/organize dados para diversas aplicações e satisfaça a todas estas. O ABD deve saber lidar com requisitos conflitantes de diversas aplicações a fim de obter um melhor esquema físico.

### 3.1.1. Índices

A utilização de índices nos campos de uma tabela, que freqüentemente está associada a restrições de consultas baseadas nestes campos, pode melhorar o desempenho destas consultas. Por outro lado, existem casos, em que mesmo havendo índices para serem utilizados, é mais eficiente executar uma varredura completa na tabela. Por isso, devemos observar a razão entre os registros selecionados e a quantidade total de registros presente na tabela. Em [Niemec03], aconselha-se que esta razão para a utilização de índices seja de até 5 %, e acima disto, é sugerido a realização de um acesso do tipo *full scan* na tabela. Porém, a razão adequada pode variar de acordo com a evolução dos dados existentes na tabela.

A existência de índices em uma tabela também pode diminuir o tempo de resposta das atualizações efetuadas sobre a mesma, devido à necessidade de atualização do índice. Desta forma, deve-se procurar apenas utilizar índices que realmente sejam utilizados e em tabelas na quais a importância do desempenho das consultas seja maior que a das atualizações.

A utilização de funções sobre os campos indexados na cláusula *WHERE* faz com que o *Oracle 9i* não utilize o índice existente, devido ao fato de que as funções podem alterar os valores dos campos, a menos que o índice seja do tipo baseado em funções. No quadro 3.1, mostramos um exemplo de uma consulta feita sobre a tabela *Funcionario* que não utiliza um índice existente sobre o campo *data\_nascimento* por este motivo. Tal consulta poderia ser reescrita de forma que o índice fosse utilizado. Para isto, aplica-se a função inversa sobre o valor comparado com o campo, da maneira como é mostrado no quadro 3.2.

**Quadro 3.1 – Consulta utilizando função que altera campo.**

```
SELECT nome, data_nascimento
FROM Funcionario
WHERE TRUNC(data_nascimento)='13-AUG-79' ;
```

**Quadro 3.2 – Consulta utilizando função que altera valor.**

```
SELECT nome, data_nascimento
FROM Funcionario
WHERE data_nascimento=TO_DATE('13-AUG-79') ;
```

O Oracle 9i também permite a criação de índices baseados em funções. Este tipo de índice pode ser utilizado caso existam consultas que sejam restringidas utilizando-se funções, expressões aritméticas ou funções PL/SQL sobre colunas de uma tabela [Niemec03]. Isto também evita que o índice não seja utilizado por causa da existência da função. O quadro 3.3 exemplifica a criação de um índice baseado em função. A função *ADD\_MONTHS* é utilizada para somar uma determinada quantidade de meses a uma data. O índice baseado em função *func\_ind2* pode ser utilizado na consulta mostrada no quadro 3.4, que é feita sobre a tabela *Funcionario*.

**Quadro 3.3 – Criando um índice baseado em função.**

```
CREATE INDEX func_ind2 ON Funcionario
(ADD_MONTHS(data_nascimento, 2));
```

**Quadro 3.4 – Consulta que utiliza índice baseado em função.**

```
SELECT nome, data_nascimento
FROM Funcionario
WHERE ADD_MONTHS(data_nascimento, 2)=TO_DATE('13-AUG-79') ;
```

A escolha da criação ou utilização de um índice deve ser efetuada de acordo com a seletividade do índice. Um índice é mais seletivo quando as colunas que o compõem possuem uma menor quantidade de valores repetidos. Um exemplo de índice bastante seletivo é aquele criado sobre uma chave primária ou candidata de uma tabela.

Existem diversas formas de calcular a seletividade de um índice. Para isto, deve-se analisar a tabela ou o índice em questão. Uma forma de realizar este cálculo é através da

obtenção da razão entre o número de registros da tabela e a quantidade de valores distintos existentes na mesma. Para isto, podemos consultar o valor da coluna *distinct\_keys* usando a visão *USER\_INDEXES* existente no SGBD *Oracle 9i* (ver seção 2.4). Quanto maior for a seletividade do índice, menor será a quantidade de registros retornados por esta consulta e maiores serão as chances de utilização do índice.

Índices concatenados ou compostos são índices criados sobre múltiplas colunas. Para alguns SGBD, o desempenho na execução de consultas difere quando se utiliza índices concatenados, ao invés de usar múltiplos índices sobre cada coluna existente no índice concatenado. Certos SGBD utilizam um índice concatenado existente apenas quando as colunas seletivas na cláusula *WHERE* usam todos os atributos existentes no índice. Outros, utilizam o índice quando um subconjunto das colunas iniciais do índice composto é usado para restringir a consulta.

Existem diversos tipos de índices disponíveis no *Oracle 9i*. Cada tipo é mais adequado para ser usado em uma ocasião específica. A seguir, listaremos os principais índices utilizados [Niemec03], e explicaremos qual o melhor momento de usá-los:

- Índices *b-tree* – é o tipo de índice padrão utilizado no *Oracle 9i*. Utilizam a estrutura de árvore B, de forma que cada ponteiro para os registros são encontrados com um número de acessos igual ao nível de profundidade da árvore. Além disso, cada ponteiro referencia o ponteiro de valor anterior e o de valor posterior, obtendo um ganho significativo em consultas que requerem ordenação. O *Oracle 9i* não indexa registros contendo o valor *NULL* nos campos indexados;
- Índices *bitmap* – criam um *bitmap* com os valores dos campos e os *ROWIDs* dos registros associados. Podem, por isto, ocupar muito espaço quando existe uma grande quantidade de valores distintos nas colunas indexadas. Este tipo de índice é uma exceção à regra geral, pois as consultas utilizando índices *bitmap* podem retornar grande parte dos registros de uma tabela. São ideais para consultas *OLAP (Online Analytical Processing)* [Niemec03] e para campos com poucos valores distintos. As atualizações contendo valores já existentes nas tabelas, e feitas com o auxílio destes índices, são bastante eficientes quando

comparadas com o uso de outros tipos de índices. Porém, elas são bastante custosas quando novos valores são inseridos, por causa da necessidade de atualização do *bitmap* para cada *rowid* existente;

- Índices *hash* – utiliza uma função de *hashing* definida sobre o valor do campo indexado que calcula uma chave *hash* associada a uma linha da tabela *hash*. Cada linha da tabela *hash* possui um *cluster* associado com uma chave de *cluster* definida, onde todos os registros associados à chave de *cluster* são inseridos em um mesmo bloco. É uma das formas mais rápidas de acesso aos dados, já que tendem a organizar os registros relacionados dentro de um mesmo bloco necessitando assim de poucas operações de Entrada e Saída (E/S). Porém, apesar das vantagens, é necessário conhecer o número de valores distintos para as colunas indexadas, para que não hajam colisões com várias chaves de *cluster* associadas a uma única chave de *hash* e não sejam assim, criados blocos de estouro, necessitando de mais E/S. As tabelas *hash* geralmente são superestimadas para evitar colisões, e com isto, tendem a utilizar espaço extra, deixando mais lentas as varreduras completas sobre as mesmas;
- Tabelas Organizadas por Índice (TOI) – os registros destas tabelas são armazenados na ordem da chave primária e possuem a mesma estrutura de armazenamento de um índice *b-tree*. Possuem índices do tipo *cluster*, nos quais os dados têm a mesma ordem que os índices. Assim, as tabelas organizadas por índice possuem acessos mais rápidos no uso de consultas exatas e de intervalos sobre os valores das chaves primárias;
- Índices de chave reversa – às vezes, uma parte do índice é bem mais utilizada do que outras através de acessos seqüenciais, gerando uma concorrência maior sobre uma parte do disco. Para resolver este problema, podemos reverter os valores dos índices armazenados de forma que valores seqüenciais, tais como 4567, 4568 e 4569 sejam armazenados como 7654, 8654 e 9654, passando a ocupar geralmente blocos diferentes e diminuindo o acesso sobre apenas uma parte do disco. Alguns índices como o *bitmap* e o TOI não podem ser revertidos;
- Índices baseados em função – são usados para que as consultas que utilizam

funções ou expressões sobre colunas indexadas possam usufruir do índice, como foi visto anteriormente nesta seção;

- Índices particionados – pode-se dividir um índice existente em índices menores para possivelmente serem acessados em menos tempo. Além disso, os índices particionados podem ser gravados em diferentes partições ou discos, aumentando a execução paralela das operações.

### 3.1.2. Partições

Tabelas de dados e índices podem ser decompostos em objetos menores através da criação de partições. As partições são varridas mais rapidamente, por se tratar de partes menores da entidade original. O aumento do paralelismo pode ser obtido também a partir da criação de partições, que diminuem a disputa por determinada entidade quando são requisitadas partes diferentes da mesma. As partes de uma mesma entidade podem estar armazenadas em discos diferentes, aumentando ainda mais o paralelismo.

O *Oracle 9i* permite criar subpartições que são partições sobre uma partição existente na base de dados. Além disso, permite também que cada partição possua uma estrutura de armazenamento diferente. As partições geram mais opções para que o SGBD possa otimizar as consultas sobre a entidade particionada. O ABD deve analisar qual deve ser o nível de particionamento adequado para cada entidade.

As partições podem ser criadas sobre simples ou múltiplas colunas em uma tabela e podem ser divididas de acordo com faixas de valores, valores em listas e algoritmos *hashing* sobre colunas. Várias ações podem ser executadas sobre as partições, dentre elas:

- Partições podem ser adicionadas e removidas sem afetar as outras partições de uma tabela;
- Uma partição pode ser transformada em duas ou mais partições e vice-versa;
- Partições individuais podem ser renomeadas; e
- As partições podem ser movidas para diferentes *tablespaces*.

O nível de particionamento e a escolha dos campos, sobre os quais a entidade deverá ser particionada, estão entre os principais problemas existentes na criação das partições. Deve-se observar se partes das tabelas são geralmente consultadas de forma separada de acordo com determinados valores dos campos. Caso haja consultas freqüentes restringidas sobre determinadas colunas, é provável que o particionamento deva ser feito sobre estas colunas e os valores geralmente utilizados podem definir a melhor escolha na divisão por faixas ou listas de valores.

Além disso, deve-se observar se o particionamento é realmente a melhor opção. Apesar das vantagens apresentadas, a criação de partições pode gerar um modelo físico mais complexo de ser mantido e *overheads* adicionais. Os mesmos motivos também devem ser levados em consideração na decisão sobre o particionamento dos índices.

### **3.1.3. Tipos dos Campos**

Em geral, diversos tipos de dados podem ser utilizados para definir um determinado campo de uma tabela. Os tipos podem ter tamanhos variáveis ou fixos, e alguns ainda podem ter o tamanho especificado pelo projetista da aplicação. Todavia, nem todos os tipos possíveis são adequados. Veremos abaixo como isto pode influenciar no desempenho.

Deve-se observar os tipos de dados para que se utilize o tipo adequado. Uma escolha inadequada pode desperdiçar espaço em uma tabela. A princípio, o custo de armazenamento tem sido relativamente barato e alguns *bytes* a mais em um registro podem não parecer muito. Contudo, o espaço adicional pode ser grande o suficiente, caso a quantidade de registros existentes na relação seja alta. Deste modo, a quantidade de memória excedente pode influenciar no tempo de resposta das consultas.

A precisão dos tipos de dados também deve ser observada para que não ocorra perda de informação devido a uma tentativa de melhoramento na performance. Por outro lado, também pode ocorrer o fato de se estar utilizando um tipo que possui alta precisão, mas esta escolha pode ter sido feita sem necessidade ou consciência de suas implicações.

Se os valores de um atributo variam bastante e ocorrem poucas atualizações, deve-se considerar o uso de campos de tamanho variável. Assim, pode-se obter uma melhor utilização do espaço. Porém, se existirem muitas atualizações, então deve-se levar em consideração o uso de campos de tamanho fixo, por causa do custo decorrente do gerenciamento dos dados de campos variáveis.

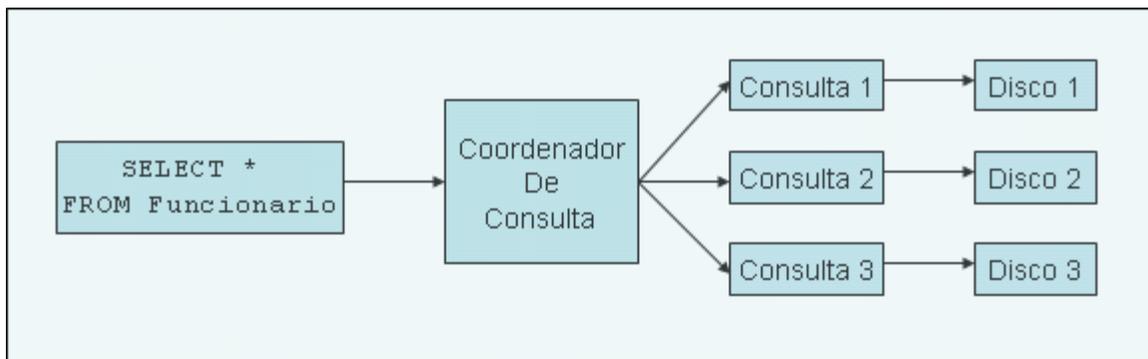
Alguns tipos de dados possuem tamanhos que podem variar até 4 *gigabytes*. O *Binary Large Object* (BLOB) é um exemplo destes tipos. Devido ao tamanho destes campos, é aconselhado o armazenamento destes tipos de dados em discos separados para permitir a leitura seqüencial dos trechos não-BLOB da tabela [Sasha03].

### **3.1.4. Paralelismo**

Os trabalhos desenvolvidos para obter melhoramento de performance através da execução paralela não é muito recente, como pode-se ver em [Bultzingsloewen89]. Os servidores de bancos de dados geralmente possuem vários processadores e discos para otimização de performance. Em servidores com apenas um único processador e um único disco, são usadas técnicas que fazem um pseudo-paralelismo como *pipelining* [Patterson98], na qual partes diferentes de uma mesma CPU são utilizadas ao mesmo tempo por instruções distintas.

É fácil observar que o paralelismo possui maior efeito quando várias consultas e atualizações sobre o banco de dados estão sendo executadas ao mesmo tempo. Porém, uma única consulta também pode gerar um plano de acesso no qual suas operações devam ser executadas em paralelo.

O *grau de paralelismo* de uma tabela é o número de processos que podem ser usados ao mesmo tempo para a execução de uma consulta sobre esta tabela. Na figura 3.1, pode-se observar um plano de execução de uma consulta utilizando paralelismo. Esta consulta executa uma varredura completa na tabela *Funcionario* utilizando três processos que são executados paralelamente para atingir o resultado final da consulta, e por isso, o grau de paralelismo desta consulta é três.



**Figura 3.1 - Consulta executada paralelamente (modificada de [Niemec03]).**

Podemos ver ainda na figura 3.1, que existe um processo adicional chamado *coordenador de consulta*, que é responsável por gerenciar as consultas que são executadas paralelamente. Observe ainda que os processos estão acessando discos separados, para fins de otimização de modo que não haja gargalos de E/S.

O *Oracle 9i* possui suporte a operações paralelas dos tipos DDL e DML. Vários fatores podem influenciar no desempenho de consultas paralelas. O ABD e o desenvolvedor da aplicação devem estar cientes dos fatores limitadores [Niemec03], tais como:

- A quantidade de processadores existentes no servidor de bando de dados;
- O número de discos disponíveis no servidor de banco de dados;
- O limite do grau de paralelismo aceito no perfil de usuário;
- A existência de partições sobre a entidade na qual se deseja efetuar uma consulta paralela;
- A alocação das partições em discos separados;
- A quantidade de processos paralelos de outros usuários;
- O número de processos paralelos de outros usuários sobre a mesma entidade;
- e
- A configuração de parâmetros tais como: `PARALLEL_MAX_SERVERS` (define o número máximo de processos paralelos no servidor para a instância *Oracle*), `PARALLEL_ADAPTIVE_MULTI_USER` (utilizado para definir

se o *Oracle 9i* deve se adaptar de acordo com a quantidade de usuários), `PARALLEL_AUTOMATIC_TUNING` (permite que a sintonia da execução paralela de consultas seja efetuada de forma automática) [Niemec03].

### 3.1.5. Normalização x Desnormalização

A *normalização* é um processo utilizado para remover anomalias e duplicações em bases de dados que utilizam o modelo relacional [Powell04]. A normalização utiliza uma série de conjuntos de regras chamadas *formas normais*. Estas regras geralmente funcionam dividindo uma relação em duas ou mais relações menores associadas por chaves.

As diferentes formas normais geram vários níveis de camadas de normalizações, de modo que cada forma normal é o refinamento da forma normal anterior. A normalização utiliza o conceito de dependência funcional, conforme visto na seção 2.3.2, de modo que cada coluna não chave deve ser funcionalmente dependente exclusivamente da chave primária da relação. Por diminuir a duplicação de dados, a normalização minimiza o espaço de armazenamento dos dados.

As relações desnormalizadas apresentam diversos problemas. A atualização do valor de uma coluna pode gerar a necessidade de alteração de vários registros que apresentam este mesmo valor, podendo haver inconsistência caso estes não sejam atualizados completamente. Além disso, é fácil observar que a alteração de vários registros consome muito mais tempo que a atualização de apenas um registro. As remoções de registros também podem excluir informações de forma indesejável, e a inclusão de novos registros pode aumentar ainda mais a duplicação dos dados.

As consultas sobre tabelas normalizadas apresentam melhor performance quando todas as colunas necessárias estão presentes na tabela normalizada. Isto deve-se ao fato de que as tabelas normalizadas possuem uma menor *cardinalidade* (número de campos presentes em uma relação) e desta forma, é necessário percorrer uma menor quantidade de *bytes* para se obter os dados desejados. Além disso, tabelas normalizadas geralmente evitam o uso da cláusula *DISTINCT*, pois evitam que os dados sejam duplicados em cada coluna.

Apesar das vantagens da normalização, em alguns casos pode ser necessário desnormalizar uma relação para obter uma melhor performance. Algumas consultas podem requerer a junção de várias tabelas normalizadas, e a operação de junção de tabelas é relativamente cara. Consultas sobre tabelas desnormalizadas tendem a ser mais eficientes, porém as alterações sobre dados destas tabelas costumam ser mais demoradas.

Antes de tomar uma decisão sobre o nível desejado de normalização ou desnormalização de relações, o ABD deve analisar as frequências das consultas e atualizações submetidas sobre as tabelas. Isto pode ser decisivo no momento da escolha, pois caso uma relação seja freqüentemente atualizada, então a desnormalização pode gerar problemas de desempenho.

### **3.1.6. Cache de Tabelas**

No hardware de um sistema computacional existem vários tipos de memórias, que são geralmente classificadas de acordo com o seu nível possível de desempenho. Quanto maior for o desempenho de uma memória, maior é o seu custo. O ideal seria que todos os sistemas utilizassem apenas as memórias mais rápidas, porém o custo destas inviabiliza esta opção.

Para obter uma grande capacidade de armazenamento e velocidades próximas das memórias de melhor desempenho, os sistemas computacionais utilizam hierarquias de memórias. Além disso, são utilizados o princípio da localidade espacial e o princípio da localidade temporal. O *princípio da localidade espacial* afirma que dados situados próximos são geralmente acessados juntos. O *princípio da localidade temporal* diz que dados acessados recentemente possuem uma maior probabilidade de serem acessados novamente.

Os SGBD também utilizam hierarquias de memórias, nas quais os dados acessados freqüentemente são armazenados em *CACHE* que é uma memória mais eficiente. O ABD pode influenciar no armazenamento de dados em *CACHE* informando ao SGBD que uma determinada tabela deve ser armazenada lá, aumentando assim, o desempenho das

consultas efetuadas sobre esta tabela.

Geralmente, nem todas as tabelas podem ser armazenadas no mais alto nível de hierarquia de memória. Para isto, deve-se analisar o tamanho disponível para o *buffer pool*, o tamanho das tabelas que se deseja armazenar e a importância da eficiência de consultas sobre estas tabelas para o sistema de banco de dados. Para saber como indicar ao SGBD na criação de uma tabela, que esta deve ser armazenada em *CACHE* ver seção 2.4.

### **3.1.7. Clusters**

Um *cluster* é um tipo de estrutura usada no armazenamento de tabelas que faz com que um grupo de tabelas compartilhem os mesmos blocos de dados. As tabelas são agrupadas porque possuem colunas compartilhadas, em comum e que são normalmente utilizadas juntas. Por exemplo, as tabelas *Funcionario* e *Departamento* podem possuir um campo que representa o número do departamento, e este campo pode ser compartilhado entre as duas tabelas a partir desta estrutura [Baylis01].

Quando duas tabelas são criadas em um *cluster*, as operações de E/S são minimizadas, aumentando assim a eficiência de junções. Após realizar a criação de um *cluster*, é aconselhado criar um índice sobre o mesmo. Além disso, não se deve criar tabelas sobre esta forma de armazenamento que sejam normalmente consultadas separadamente. Na seção 2.4, é mostrado como criar um *cluster* em um dado esquema físico.

### **3.1.8. Esquemas Distribuídos**

Um projeto de base de dados distribuída inclui decisões sobre a localização de dados entre os sites. Para isto, deve-se saber a localização dos SGBD e dos aplicativos que os utilizam. O projeto de esquemas distribuídos requer os mesmos cuidados do desenvolvimento de esquemas centralizados, porém existe uma preocupação adicional que é a forma como os dados serão distribuídos.

Uma das razões para a distribuição dos dados é a possibilidade de decomposição de uma entidade em fragmentos, de forma que cada um deles seja tratado como uma unidade, aumentando o acesso concorrente aos dados. Uma consulta pode ser resolvida através de várias sub-consultas sobre dados localizados em sites diferentes, aumentando assim o desempenho da execução de consultas [Ozsu99].

Em Bancos de Dados Distribuídos (BDD), é desejável que os dados possam ser fragmentados em partes mutuamente exclusivas, de forma que um site raramente necessite de informações contidas em outro site. Caso os fragmentos possuam partes em comum, pode ser necessário que ambos fragmentos estejam atualizados, gerando assim um custo adicional.

Outro problema em BDD é quando são necessários dados que se encontram em outro site. O custo de transmissão de dados pela rede normalmente é bem superior ao tempo de processamento destes dados. Deve-se por isso tentar minimizar a quantidade de dados que trafegam pela rede.

Antes dos dados serem distribuídos, existe a necessidade de escolher o grau de fragmentação que o banco de dados deverá sofrer. Esta escolha é essencial para a eficiência das consultas, pois determinará o nível de concorrência a ser obtido e a quantidade de dados a serem trafegados pela rede.

São muitos os fatores que podem influenciar no desenvolvimento de um bom esquema físico distribuído. Nem sempre a distribuição é a melhor solução, deve-se mensurar os possíveis ganhos e perdas de acordo com estatísticas sobre os dados armazenados e consultas submetidas. Além disso, os fragmentos devem estar localizados de acordo com os locais que mais os referenciam.

## 3.2. Consulta

Para execução de uma consulta, normalmente existem vários planos de consulta diferentes que retornam o mesmo resultado. O otimizador é responsável por escolher um plano que possua o menor custo. Nem sempre é possível escolher o melhor plano de acesso,

pois a quantidade de planos possíveis pode ser enorme, não permitindo ao otimizador, verificar todas possibilidades.

O *Oracle 9i* permite a utilização de dois tipos de estratégias de otimização: (i) a otimização baseada em regras e (ii) a otimização baseada em custos. Na *otimização baseada em regras*, o otimizador leva em consideração um conjunto de regras para identificar o melhor plano de consulta. Estas regras se baseiam nos caminhos de acesso possíveis, e a escolha do plano ideal/ótimo não depende de estatísticas existentes em uma base de dados particular. Na *otimização baseada em custos*, é obtido um conjunto de planos de consultas potenciais, e então, são calculados os custos de cada plano deste conjunto a partir de estatísticas armazenadas sobre a base de dados. Assim, o otimizador baseado em custos é normalmente a melhor estratégia de otimização para as consultas sobre o banco de dados.

A forma como as consultas são escritas influenciam a escolha do otimizador. Nas próximas seções, veremos de que forma as consultas podem ser reescritas para obter um melhor desempenho.

### 3.2.1. Operadores de Ordenação

A ordenação é uma operação de alto custo que é geralmente efetuada com bastante frequência. Por esta razão, deve-se saber quando seu uso pode ser evitado. Uma ordenação é executada explicitamente através da cláusula *ORDER BY*. Além de executar operações de ordenação de forma explícita, o *Oracle 9i* também ordena campos de uma relação de modo implícito, quando são utilizadas outras cláusulas.

O conhecimento destas cláusulas é essencial para evitar trabalho desnecessário realizado pelo SGBD. Em alguns casos, o SGBD pode determinar que não é necessário fazer uma nova ordenação, porém em outros, pode não ser tão simples determinar isto. A seguir, são listados os tipos de ordenação executados pelo *Oracle 9i* [Powell04]:

- *SORT UNIQUE* – usado para gerar uma lista de valores únicos ou quando existe

o uso da cláusula *DISTINCT*;

- *SORT ORDER BY* – utilizado para forçar uma ordenação explícita através da cláusula *ORDER BY*;
- *SORT GROUP BY* – uma ordenação é requerida para agrupar os valores duplicados em uma coluna quando se utiliza a cláusula *GROUP BY*;
- *SORT JOIN* – esta ordenação é executada sobre as tabelas, antes de uma junção do tipo *SORT MERGE JOIN*; e
- *SORT AGGREGATE* – utilizado quando uma agregação é feita sobre muitas linhas.

Deve-se saber antes da execução de uma ordenação, se os registros já estão ordenados na forma desejada. Por exemplo, em uma tabela com tipo de índice TOI, os dados se encontram ordenados pela chave primária da relação. Com base no conhecimento da ordenação dos dados, pode-se escolher reescrever uma consulta para obter um melhor desempenho. A escolha de uma junção do tipo *SORT MERGE JOIN* pode ser mais adequada se os dados já estiverem ordenados.

O ABD pode verificar as estatísticas sobre as quantidades de ordenações efetuadas na memória ou em disco. Estas estatísticas variam de aplicação para aplicação de acordo com o número de ordenações necessárias, e além disso, dependem também da quantidade de dados a serem ordenados em cada ordenação. Quanto maior o número de ordenações, maior é a necessidade de operações de E/S.

Para se obter estatísticas sobre quando geralmente ocorrem as ordenações, podemos consultar a visão *V\$SYSSTAT* do sistema como é mostrado no quadro 3.5.

**Quadro 3.5 – Exibindo estatísticas sobre ordenação.**

```
SELECT name, value
FROM V$SYSSTAT
WHERE name LIKE 'sorts (memory)'
      OR name LIKE 'sorts (disk)';
```

O parâmetro de inicialização *SORT\_AREA\_SIZE* define o tamanho da memória utilizada para ordenação. Se existem muitas ordenações efetuadas sobre o disco pode ser

necessário incrementar este parâmetro para obter uma melhor performance, dado que as ordenações feitas em memória são bem mais rápidas que as ordenações realizadas em disco. Além do aumento do valor de *SORT\_AREA\_SIZE*, pode-se desejar incrementar, também o valor do parâmetro *SORT\_AREA\_RETAINED\_SIZE* que indica a quantidade de área de memória disponível/reservada para armazenar os valores das ordenações efetuadas para usos posteriores [Whalen96].

### 3.2.2. Junções

Uma operação de junção sobre duas tabelas gera como resultado um conjunto que possui todos os elementos da primeira tabela concatenados com todos elementos da segunda tabela que obedece a uma restrição indicada na consulta. A operação de junção é comutativa, ou seja a ordem dos fatores na junção não altera o resultado [Fernandes02]. A junção costuma ser uma operação cara, porém necessária para evitar o uso de produtos cartesianos que requerem mais recursos do sistema.

Em uma junção feita no *Oracle 9i*, há dois tipos de tabelas. A primeira delas é normalmente conhecida como *tabela externa* ou *controladora*, sendo a partir desta que o *Oracle 9i* toma decisões sobre as junções. A segunda tabela é chamada de *tabela interna* [Niemec03].

A ordem em que as tabelas são dispostas na cláusula *FROM* pode influenciar na escolha da melhor solução. Pode parecer simples escolher a ordem das tabelas, quando existem apenas duas ou três delas na consulta. Contudo, quando se trata da junção de oito tabelas, torna-se bastante difícil encontrar a melhor ordem, pois trata-se de 40320 possíveis combinações<sup>1</sup>.

A junção de tabelas é uma operação binária, ou seja, quando ocorrem junções de três ou mais tabelas, essas operações são feitas duas a duas. Existem vários tipos de junções, e cada uma delas apresenta um algoritmo diferente que é mais adequado para uma ocasião específica. Explicaremos a seguir cada um destes tipos.

---

<sup>1</sup> Para calcular o número de combinações possíveis, utilizamos a fórmula de permutações ( $P(n)=n!$ ).

### **3.2.2.1. Junção *NESTED LOOPS***

Uma junção do tipo *NESTED LOOPS* é realizada lendo-se todos registros da tabela controladora, e para cada registro da tabela controladora, são lidos todos os registros da tabela interna a fim de encontrar combinações válidas para a junção. À primeira vista, este tipo de junção apresenta um custo semelhante aos produtos cartesianos. Todavia, ele possui a vantagem, sobre os outros tipos, de apresentar mais rapidamente os primeiros resultados obtidos. Quando os dados são exibidos parcialmente ao longo de telas de consultas, é possível obter os primeiros registros para visualização, enquanto são buscados outros registros.

Outras ocasiões ideais para a utilização de uma junção *NESTED LOOPS* são na existência de um índice bastante seletivo sobre a tabela interna e na possibilidade de utilização de uma tabela externa pequena. Caso não existam índices seletivos sobre a tabela interna, este tipo de junção pode ser bastante ineficiente [Niemec03].

### **3.2.2.2. Junção *SORT-MERGE***

As junções do tipo *SORT-MERGE* ordenam a primeira e a segunda tabela pelos seus respectivos campos de junção. Depois que as tabelas estão ordenadas, torna-se mais fácil efetuar a combinação das tuplas que satisfazem a condição de junção.

A combinação dos registros é feita utilizando dois ponteiros, um para cada tabela. Os ponteiros inicialmente apontam para o primeiro registro de cada tabela ordenada. Caso os registros satisfaçam a condição de junção, a combinação deles é acrescentada ao conjunto resultante da operação. Quando os ponteiros apontam para registros que não satisfazem a condição, o ponteiro que aponta para o registro de menor ou igual valor da coluna de junção é incrementado. Esta operação continua até que uma das tabelas seja completamente percorrida [Niemec03].

Este tipo de junção é normalmente mais eficiente que a junção *NESTED LOOPS* quando não existem índices bastante seletivos sobre a coluna de junção de uma das tabelas

ou quando as duas tabelas são suficientemente grandes. Porém, a junção *MERGE-SORT* só pode ser usada quando são utilizadas *equijunções*, ou seja, quando são utilizadas condições de igualdade sobre os campos de junção.

### **3.2.2.3. Junção CLUSTER**

Para que haja uma junção do tipo *CLUSTER*, é necessário que as duas tabelas envolvidas na junção façam parte de um mesmo *CLUSTER*, e que exista um índice sobre o mesmo. Além disto, este tipo de junção só é válido para *equijunções* sobre as chaves de cluster das tabelas. Esta junção é um tipo especial de junção *NESTED LOOPS* que possui várias vantagens. Uma delas é a existência de um índice sobre ambas tabelas. As tabelas de junção também compartilham os mesmos blocos e estão ordenadas previamente de acordo com as colunas de junção. Por estes fatos, as junções *CLUSTERS* geralmente possuem excelentes desempenhos [Niemec03].

### **3.2.2.4. Junção HASH**

Em junções *HASH*, o *Oracle 9i* escolhe uma das tabelas para ser a *tabela de hash*, como um índice temporário, sobre os campos de junção da tabela escolhida (tabela controladora). Normalmente, o SGBD seleciona a menor delas, por apresentar um menor custo de criação para a tabela de hash. Uma varredura completa é efetuada sobre a outra tabela, enquanto a tabela de hash é utilizada para encontrar combinações entre elas.

Para que este tipo de junção ocorra de maneira eficiente, é necessário que haja espaço suficiente na memória para a criação da tabela de hash. É mais eficiente que uma junção *NESTED LOOPS* caso não existam índices muito seletivos sobre os campos de junção. Dependendo do custo de ordenação para as duas tabelas, esta última ou a junção *HASH* pode ser mais rápida que uma junção *SORT-MERGE*.

A junção *HASH* é outro tipo que funciona apenas para *equijunções*. Outra desvantagem é que pode haver gargalo nas operações de E/S, caso não haja espaço suficiente para a criação da tabela de hash em memória [Niemec03].

### 3.2.2.5. *Junção de índices*

A junção de índices é um tipo especial de junção *HASH* que é feita sobre índices, e efetuada quando toda informação necessária está presente nos índices. Este tipo de junção é bastante eficiente, porque as estruturas dos índices geralmente ocupam menos espaço que as tabelas. Isto ocorre principalmente, quando as tabelas apresentam muitos campos ou quando algumas colunas possuem tamanhos relativamente grandes.

Pode ser necessário incluir alguns índices em colunas não indexadas, para que seja possível a utilização deste tipo de junção. As varreduras completas e rápidas diferem da junção de índices por utilizarem apenas um índice composto ao invés de vários índices, sendo cada um deles para cada coluna de junção. Geralmente, ambos são bastante eficientes [Niemec03].

### 3.2.3. **Sub-consultas**

As sub-consultas são consultas executadas dentro de outras consultas. Elas são consideradas consultas complexas, e geralmente podem ser reescritas de diversas formas. As sub-consultas podem ser divididas em dois tipos [Lifschitz04]: correlacionadas e não-correlacionadas. Nas sub-consultas correlacionadas, a consulta interna referencia algum campo da consulta externa. Nas sub-consultas não-correlacionadas, cada uma das consultas pode ser executada independentemente por não haver referências entre elas, caso a consulta interna seja substituída por um conjunto de tuplas.

Podemos ver no quadro 3.6-a, um exemplo de uma sub-consulta correlacionada, na qual a consulta interna referencia o campo *d.id\_departamento* da consulta externa. No quadro 3.6-b, é mostrado um exemplo de uma consulta não correlacionada.

**Quadro 3.6 – Exemplos de tipos de sub-consultas.**

```
SELECT nome
FROM Departamento d
WHERE EXISTS
  (SELECT nome
   FROM Funcionario f
   WHERE f.salario > 1000
        AND f.id_departamento = d.id_departamento);
```

(a)

```
SELECT MAX(salario)
FROM Funcionario
WHERE id_departamento IN
  (SELECT DISTINCT id_departamento
   FROM Departamento);
```

(b)

A maioria das sub-consultas são transformadas pelo otimizador do *Oracle 9i* em junções correspondentes, principalmente quando existe um índice sobre uma das tabelas. Em seguida, um dos métodos de junção é escolhido. Observe que o quadro 3.6 ilustra a utilização dos operadores *IN* e *EXISTS*, os quais são bastante comuns no uso de sub-consultas. No que segue, explicaremos quando é mais adequado utilizar cada um deles.

Quando uma sub-consulta que utiliza o operador *IN* não é convertida diretamente em uma junção, o *Oracle 9i* executa a consulta interna e armazena o resultado em uma tabela temporária. Em seguida, uma junção do tipo *SORT-MERGE* ou *HASH* normalmente é realizada entre a tabela externa e a tabela temporária, pois não existe um índice referenciando a tabela temporária [Silva03].

O operador *EXISTS* retorna um valor booleano. Ele retorna *FALSE* caso não haja nenhum registro encontrado na consulta interna e *TRUE*, caso tenha encontrado pelo menos um registro. Se um registro for encontrado antes do final da execução da consulta interna, o resultado é retornado sem que este operador precise varrer o restante das tuplas da tabela correspondente.

De acordo com as características do operador *EXISTS*, podemos observar que sua execução tende a ser mais eficiente que o operador *IN* em sub-consultas não-correlacionadas. Além disso, a existência de um índice, sobre campos que restringem a consulta interna, pode fazer com que o resultado da sub-consulta utilizando *EXISTS* seja

praticamente imediato.

Em sub-consultas correlacionadas utilizando *EXISTS*, podem ser requeridas várias execuções para estes mesmos operadores, caso a sub-consulta não seja transformada em uma junção. Quando isto acontece, algumas vezes é possível transformar a sub-consulta correlacionada em uma sub-consulta não-correlacionada utilizando o operador *IN*. Esta sub-consulta necessita apenas de uma execução da consulta interna, podendo assim ser mais eficiente.

### 3.2.4. Hints

O otimizador de consultas é um sofisticado componente de software que pode considerar muitos fatores e criar planos de consulta muito eficientes. Porém, existem situações em que o otimizador não consegue descobrir o plano de consulta mais eficiente [England01]. Alguns SGBD, como o *Oracle 9i*, possuem um mecanismo chamado *hint*, que permite ao desenvolvedor ou ao ABD, influenciar na escolha de uma técnica particular para ser executada no plano de consulta. Alguns *hints* foram escolhidos para serem mostrados. Esta escolha se baseou de modo a exibir *hints* que permitam indicar sugestões descritas neste capítulo.

*Alias* pode ser definido como “apelidos” utilizados para renomear um item em um comando SQL. Uma observação importante é que o uso deles deve ser feito com cuidado. Quando *alias* são usados em itens e o comando SQL utiliza *hints*, deve-se referenciar estes itens nos *hints* através dos *alias*. Caso contrário, os *hints* serão ignorados. A seguir, será explicado o uso de alguns *hints* utilizados para sugerir a escolha de um caminho de acesso específico.

#### 3.2.4.1. CHOOSE

Para usar uma estratégia de otimização baseada em custo, pode-se modificar o parâmetro de inicialização *OPTIMIZER\_MODE* com o valor *CHOOSE*. Esta modificação

altera a estratégia de otimização para todas as consultas. Outra forma é utilizar o *hint CHOOSE*. Contudo, esta opção só pode ser usada após as tabelas envolvidas na consulta serem analisadas através do pacote DBMS\_STATS ou comando ANALYSE [Powell04], gerando assim estatísticas sobre estas tabelas. Caso contrário, será utilizada uma estratégia baseada em regras. O quadro 3.7 mostra o uso do *hint CHOOSE*.

**Quadro 3.7 – Usando o hint CHOOSE.**

```
SELECT /*+ CHOOSE */ nome, data_nascimento  
FROM Funcionario  
WHERE salario > 1000;
```

#### 3.2.4.2. *RULE*

A fim de utilizar uma estratégia baseada em regras, pode-se atribuir ao parâmetro OPTIMIER\_MODE, o valor *RULE*. Desta forma, a estratégia de todas consultas que forem otimizadas é afetada. Pode-se também usar o *hint RULE*, para que o SGBD utilize sua base de regras. O uso do *hint RULE* faz com que o SGBD evite usar os outros *hints*, menos *DRIVING\_SITE* e *ORDERED* [Niemec03]. É mostrado no quadro 3.8, um exemplo de utilização do *hint RULE*.

**Quadro 3.8 – Usando o hint RULE.**

```
SELECT /*+ RULE */ nome, data_nascimento  
FROM Funcionario  
WHERE salario > 1000;
```

#### 3.2.4.3. *FULL*

O *hint FULL* é utilizado para obter uma varredura completa em uma tabela, mesmo que haja um índice sobre a mesma. Este *hint* pode ser bastante útil quando um grande percentual das tuplas de uma tabela for retornado ou quando o índice existente não for bastante seletivo, melhorando assim a performance da consulta. Por exemplo, em uma dada

aplicação de BD poderíamos ter uma consulta sobre todos funcionários com salário superior a 1000 (ver quadro 3.9).

**Quadro 3.9 – Usando o hint FULL.**

```
SELECT /*+ FULL(Funcionario) */ nome, data_nascimento
FROM Funcionario
WHERE salario > 1000;
```

#### **3.2.4.4. INDEX**

Pode-se optar pelo uso de um determinado índice através do *hint INDEX*. Além disto, pode existir vários índices relacionados com uma tabela e o desenvolvedor pode querer que o otimizador escolha entre apenas dois desses índices via o uso do *hint INDEX*, como é mostrado no quadro 3.10. Outra forma de uso deste *hint* é a não especificação de índices. Deste modo, o otimizador faz a escolha baseada em todos os índices existentes.

**Quadro 3.10 – Usando o hint INDEX.**

```
SELECT /*+ INDEX(Funcionario indice1, indice2) */ nome,
data_nascimento
FROM Funcionario
WHERE salario > 1000;
```

#### **3.2.4.5. ORDERED**

A ordem em que as tabelas são acessadas em uma junção influenciam no desempenho da consulta. Normalmente, é preferível que as menores tabelas ou as mais restringidas sejam acessadas primeiro. Para que as tabelas sejam acessadas na ordem em que são explicitadas na cláusula FROM, deve-se utilizar o *hint ORDERED*. O quadro 3.11 indica o uso do *hint ORDERED*.

**Quadro 3.11 – Usando o hint ORDERED.**

```
SELECT /*+ ORDERED */ f.nome, d.nome
FROM Funcionario f, Departamento d
WHERE f.id_departamento = d.id_departamento
      AND f.salario > 1000;
```

#### **3.2.4.6. DRIVING\_SITE**

Em banco de dados distribuídos, a escolha do site responsável por controlar uma determinada consulta pode afetar enormemente seu tempo de resposta. O *hint DRIVING\_SITE* pode ser usado para essa escolha, no qual definimos a tabela controladora (ver quadro 3.12), podendo a mesma estar localizada em um servidor remoto.

**Quadro 3.12 – Usando o hint DRIVING\_SITE.**

```
SELECT /*+ DRIVING_SITE(d) */ f.nome, d.nome
FROM Funcionario f, Departamento@Maceio d
WHERE f.id_departamento = d.id_departamento
      AND f.salario > 1000;
```

#### **3.2.4.7. USE\_NL**

Pode-se escolher o método de junção através do uso de *hints*. Por exemplo, o *hint USE\_NL* pode ser empregado para que uma junção do tipo *NESTED LOOPS* seja executada pelo otimizador. Este tipo de junção retorna mais rapidamente, os primeiros registros resultantes de uma consulta, como visto. Pode-se ver um exemplo de uso do *hint USE\_NL*, no quadro 3.13.

**Quadro 3.13 – Usando o hint USE\_NL.**

```
SELECT /*+ USE_NL(f d) */ f.nome, d.nome
FROM Funcionario f, Departamento d
WHERE f.id_departamento = d.id_departamento
      AND f.salario > 1000;
```

### 3.2.4.8. *PARALLEL*

Em servidores com múltiplos processadores e discos, pode-se desejar que uma consulta seja efetuada paralelamente. Usando o *hint PARALLEL*, define-se o grau de paralelismo utilizado no processamento de uma consulta sobre uma determinada relação. No quadro 3.14, é especificado que a consulta deve ser feita de modo que a tabela *Funcionario* seja varrida com grau de paralelismo igual a três.

**Quadro 3.14 – Usando o hint PARALLEL.**

```
SELECT /*+ PARALLEL(Funcionario, 3) */ nome, data_nascimento
FROM Funcionario
WHERE salario > 1000;
```

### 3.2.4.9. *NOPARALLEL*

Foi mostrado anteriormente que uma tabela pode ser criada no *Oracle 9i*, com um grau de paralelismo específico. Desta forma, todas consultas utilizadas sobre esta tabela iriam ser executadas paralelamente. Usando o *hint NOPARALLEL* (ver quadro 3.15), a consulta é efetuada sem tirar proveito de execuções paralelas.

**Quadro 3.15 – Usando o hint NOPARALLEL.**

```
SELECT /*+ NOPARALLEL(Funcionario) */ nome, data_nascimento
FROM Funcionario
WHERE salario > 1000;
```

### 3.2.4.10. *CACHE*

Uma consulta realizada sobre o banco de dados, quando os dados estão em *CACHE*, possui um menor tempo de resposta. O SGBD é responsável por utilizar políticas de gerenciamento de memória, como LRU (*Least Recently Used*). Porém, é possível utilizar o *hint CACHE* para indicar ao *Oracle 9i*, que uma determinada tabela deve ser colocada em

memória na primeira vez que for acessada, através de uma varredura completa, e em seguida, permanecendo lá se possível. No quadro 3.16, é mostrado um exemplo de uso do *hint CACHE*.

**Quadro 3.16 – Usando o hint CACHE.**

```
SELECT /*+ CACHE(Funcionario) */ *  
FROM Funcionario;
```

### 3.2.4.11. NOCACHE

Uma outra forma de influenciar no gerenciamento de memória é através da utilização do *hint NOCACHE*. Este *hint* é usado para indicar que uma determinada tabela não deve ser guardada na *CACHE* (ver quadro 3.17). Impedindo assim, através de indicações, o armazenamento de tabelas raramente acessadas ou que ocupem muito espaço, mesmo que elas tenham sido criadas com a opção *CACHE*. Observe que em um dado momento, pode-se querer ou não que a tabela esteja disponível em memória primária. E caso o desenvolvedor consiga distinguir estes momentos, ele tem a possibilidade de escolher quando uma tabela deve se mantida em *CACHE*.

**Quadro 3.17 – Usando o hint NOCACHE.**

```
SELECT /*+ NOCACHE(Funcionario) */ *  
FROM Funcionario;
```

### 3.2.5. Suprimindo índices

A existência de índices sobre uma tabela não é suficiente para que os mesmos sejam usados. Foi visto anteriormente que a seletividade do índice pode influenciar no seu uso, caso a estratégia de otimização seja baseada em custo<sup>2</sup>.

---

<sup>2</sup> Caso a estratégia seja baseada em regras, então o índice será utilizado mesmo assim.

Outras ocasiões que podem ter índices suprimidos é quando um percentual alto de registros é retornado, ou quando existe uma função sobre o campo indexado, desde que o índice sobre o mesmo não seja baseado em função. Nesta seção, será visto que o modo como uma consulta é escrita pode impedir a utilização de índices.

O uso dos operadores *IS NULL* ou *IS NOT NULL*, sobre uma coluna indexada, que verificam se o valor do campo é nulo ou não, impedem a utilização do índice. Esta supressão é devido ao fato de que o valor *NULL* é indefinido. Por isto, os registros cujas colunas possuem valores nulos não são indexados. Para se ter certeza de que um campo será totalmente indexado, pode-se utilizar as cláusulas *NOT NULL*, impedindo a inserção de valores nulos na tabela, ou *DEFAULT*, para que valores não informados não recebam valores nulos, na criação de uma tabela.

A existência do operador *NOT EQUAL* (!=) na cláusula *WHERE* também evita o uso de índices, dado que os índices não indexam uma coluna pelos valores inexistentes. Uma consulta utilizando *NOT EQUAL* pode ser reescrita para possibilitar a utilização do índice, conforme mostra o quadro 3.18. Porém, o mesmo não pode ser do tipo *HASH* ou *MERGE-SORT* que só permitem equijunções.

A aplicação de máscaras também pode impedir o uso de índices. Quando uma consulta possui uma restrição na cláusula *WHERE* utilizando uma máscara da forma *nome LIKE '%rdo'*, o uso de índices é evitado. Isto ocorre porque os caracteres iniciais não são informados, de modo que seria necessário varrer todo o índice para retornar a solução da consulta. Porém, caso uma varredura completa no índice seja mais rápida do que uma varredura completa na tabela, pode-se sugerir ao SGBD, a utilização de índices através de *hints*.

O *Oracle 9i* muitas vezes converte dados implicitamente quando existem comparações de tipos de dados divergentes. Esta conversão produz o mesmo efeito de uma função sobre o campo indexado da tabela, fazendo com que índices existentes sobre a coluna deixem de ser utilizados. Para se obter uma melhor performance com a utilização de índices, deve-se evitar então, comparações entre tipos distintos.

**Quadro 3.18 – Reescrita de consulta utilizando NOT EQUAL.**

```
SELECT nome, data_nascimento  
FROM Funcionario  
WHERE salario != 1000;
```

(a)

```
SELECT nome, data_nascimento  
FROM Funcionario  
WHERE salario < 1000  
OR salario > 1000;
```

(b)

### 3.2.6. Cláusulas e restrições duplicadas

A escrita de consultas complexas pode gerar o uso de cláusulas e restrições duplicadas. Para que isto não ocorra, deve-se sempre fazer um refinamento da consulta a fim de descobrir duplicidades que possam haver na mesma. Quanto mais se evitar estas duplicidades, mais simples e eficiente será a execução da consulta. A seguir, serão abordadas várias situações, nas quais duplicidades podem ocorrer.

A cláusula *DISTINCT* é utilizada para remover tuplas duplicadas. Seu uso acarreta na realização de forma implícita, de uma operação de ordenação. Sabe-se que as ordenações são operações custosas, e por isso, é desejável que o uso desta cláusula seja evitado sempre que possível. Ao usar *DISTINCT* deve-se primeiramente observar se algum campo (ou conjunto de campos) do tipo *UNIQUE* está sendo retornado (como por exemplo a chave primária de uma relação). Isto é suficiente para eliminar o uso da cláusula *DISTINCT*, caso não ocorram junções. Outra ocasião, corresponde ao uso em conjunto desta cláusula com o comando *GROUP BY*, que também realiza a ordenação antes de agrupar valores iguais.

O uso da cláusula *NOT NULL* para colunas, cujo esquema físico que as definem já possui esta mesma restrição, gera o custo adicional de uma varredura completa nestas tabelas. Outras duplicações ocorrem na cláusula *WHERE*, e nem sempre são fáceis de serem percebidas. Um exemplo é quando uma restrição gera uma resposta que é

subconjunto da aplicação de outra restrição, como mostra o quadro 3.19. Outro exemplo semelhante é quando várias restrições sempre geram como resultado um subconjunto de outra restrição, podendo assim a consulta ser reescrita para obter uma melhor eficiência.

**Quadro 3.19 – Utilizando restrições duplicadas.**

```
SELECT nome, data_nascimento
FROM Funcionario
WHERE estado LIKE 'PE'
      AND cidade LIKE 'Recife';
```

### **3.2.7. Consultas Distribuídas**

O processo de otimização de consultas é bem mais complexo em ambientes distribuídos. A exploração de fragmentos replicados gera um aumento no número de estratégias possíveis. Além disto, os custos de comunicação devem ser adicionados para a escolha do melhor plano de consulta.

Antes das sub-consultas serem submetidas a diversos sites, deve-se saber inicialmente, onde os dados necessários para a consulta, estão localizados. O otimizador de consultas deve selecionar os melhores lugares para processar as consultas. A permutação das ordens de execução das operações de uma consulta fornece vários planos de acesso possíveis [Ozsu99].

A velocidade de transmissão dos dados pela rede influencia na escolha da melhor opção. Os planos de consulta em redes de baixa velocidade normalmente evitam a transmissão de dados pela rede. Em redes de alta velocidade, alguns planos de consulta, inviáveis para redes de baixa velocidade, tornam-se possíveis escolhas.

### **3.2.8. WHERE x HAVING**

A cláusula *HAVING* é utilizada em agregações para restringir uma consulta, mas a

restrição é efetuada após as junções e operações de agregação. Em alguns casos, podemos substituir *HAVING* pela cláusula *WHERE* que restringe a consulta antes da operação de agregação. Assim, dados desnecessários para a consulta não sofrem ordenação devido à agregação.

Em alguns casos, é necessário utilizar *HAVING* por não ser possível o uso da cláusula *WHERE*. Exemplos desses casos são comparações de funções de agregação com valores. Porém, o uso da cláusula *HAVING* deve ser evitado sempre que possível, para que se obtenha um ganho de performance.

### 3.3. Conclusão

Para o desenvolvimento do esquema físico, deve-se ter um conhecimento prévio de estatísticas sobre a base de dados. A criação de índices é determinada pelas consultas freqüentemente enviadas ao banco de dados. A escolha do tipo de índice depende das estatísticas sobre os valores dos campos indexados.

As entidades da base de dados podem ser particionadas em entidades menores, a fim de se obter um maior processamento paralelo sobre as mesmas. A escolha do tipo de dados adequado também pode influenciar no desempenho, pois os espaços desperdiçados em um campo de uma tabela geralmente implica em uma maior quantidade de bytes lidos.

Redundâncias podem ser adicionadas, através de desnormalizações sobre dados consultados normalmente juntos, para evitar gastos de tempo decorrentes de junções. Outra forma de minimizar o tempo gasto em junções é através da criação de *clusters*. Estes são criados para que seja evitada a redundância de dados, e para que o desempenho seja otimizado devido ao fato de tabelas em um *cluster* compartilharem uma mesma estrutura.

As consultas devem maximizar o uso do esquema físico definido. Índices criados devem ser usados quando for possível obter mais eficiência, através destes. Os *hints* podem ser usados para indicar ao SGBD, a escolha de um dos possíveis caminhos a ser utilizado no plano de consulta. Deste modo, o desenvolvedor da aplicação de BD pode reescrever as consultas para influenciar na escolha feita pelo SGBD.

O SGBD proporciona vários tipos possíveis de ordenações e junções. Cada tipo é mais adequado em uma situação específica. Algumas destas operações podem ser evitadas, como por exemplo, o uso de operações duplicadas ou desnecessárias, diminuindo o tempo de resposta final.

Primeiramente, o projeto da base de dados e a reescrita de consultas caminham junto para melhorar a performance do sistema de banco de dados. Sendo assim, o SGBD normalmente escolhe a ordem em que as operações são efetuadas, através da escolha da melhor opção entre os diversos planos de acessos existentes. Finalmente, nos bancos de dados distribuídos, o tempo de comunicação entre os sites devem ser levados em consideração antes da escolha do plano de acesso.

Neste capítulo, foram vistas diversas formas de sintonia de esquemas e consultas, de modo a obter um melhor desempenho do processamento de consultas pelo SGBD. No próximo capítulo, será apresentada uma análise sobre os resultados de uma simulação, realizada durante o desenvolvimento deste trabalho.

## 4. Simulação e Análise

Primeiramente, na seção 4.1, é exibido o projeto do experimento realizado. A seção 4.2 mostra os resultados da simulação do experimento. Uma análise sobre estes resultados é apresentada na seção 4.3. Finalmente, é exibida uma conclusão sobre a simulação e análise do experimento na seção 4.4.

### 4.1. Projeto do Experimento

Nesta seção, é apresentado o projeto do experimento realizado para simulação e avaliação dos princípios de performance, levantados no desenvolvimento deste trabalho. Tal experimento teve como objetivo investigar possíveis ganhos obtidos, com o uso dos princípios mostrados no capítulo anterior. Para isto, foram analisados os planos de acesso das consultas submetidas ao banco de dados para saber os motivos da variação dos custos de cada consulta.

Para a realização do experimento, foram escolhidos o Sistema Operacional (SO) *Windows XP Professional* e o SGBD *Oracle 9i Release 2*. O hardware utilizado foi um microcomputador com CPU Athlon XP 1600+, 256 MB de memória principal e HD 60 GB 7200 RPM ATA 100. Estas escolhas devem-se ao fato de ambas ferramentas e do hardware estarem previamente disponíveis para a simulação do experimento. Além disto, o SO e o SGBD escolhidos são amplamente conhecidos e têm sido utilizados tanto na academia quanto no mercado [Niemec03, Silva03, Loney02].

Inicialmente, verificou-se a necessidade de inicializar alguns parâmetros para a configuração do *Oracle 9i* adequadamente. Estes parâmetros estão definidos no arquivo *init.ora* e podem ser redefinidos através de alteração manual deste arquivo que requer a reinicialização do SGBD, ou alguns destes parâmetros podem ser alterados dinamicamente através do comando *ALTER SYSTEM*. A redefinição dos parâmetros dinamicamente evita a reinicialização do SGBD e assim a indisponibilidade do BD.

Foi decidido utilizar a configuração padrão [Niemec03] para a maioria dos

parâmetros de inicialização, devida à restrição de memória do sistema que não permite grandes alterações. Além disto, a lista de parâmetros é bastante extensa e a alteração de alguns parâmetros pode afetar outros. Portanto, é importante reconhecer que para uma configuração mais eficiente, é preciso a realização de uma pesquisa totalmente relacionada com este foco, o que não é o tema deste trabalho. Pequenas mudanças foram efetuadas ao longo da simulação de acordo com a necessidade, e serão salientadas com a apresentação dos resultados.

Algumas ferramentas foram estudadas para a realização da simulação, tais como o *Oracle SQL Scratchpad* do *Oracle Enterprise Manager*, o *SQL\*Plus* e o *SQL\*Plus Worksheet*. Todas estas ferramentas estão incluídas no pacote de instalação do *Oracle 9i*. No quadro 4.1, podemos observar resultados de uma comparação feita entre estas ferramentas.

**Quadro 4.1 – Comparação entre ferramentas para a simulação.**

	<i>Oracle SQL Scratchpad</i>	<i>SQL*Plus Worksheet</i>	<i>SQL*Plus</i>
<i>Exibe plano de consulta</i>	SIM, de forma mais prática	SIM, através de EXPLAIN PLAN	SIM, através de EXPLAIN PLAN
<i>Salva histórico de comandos</i>	SIM	NÃO	NÃO
<i>Permite cancelar comandos</i>	SIM	NÃO	SIM
<i>Exibe tempo de execução</i>	SIM	NÃO	NÃO

Todas as ferramentas analisadas eram disponíveis e executam comandos SQL e por isso, foram consideradas como candidatas à ferramenta de suporte para realização de nosso experimento. Dentre estas, apenas o *Oracle SQL Scratchpad* exibe o plano de consulta automaticamente, e não necessita da execução da consulta (ver quadro 4.1).

De acordo com o quadro 4.1, pode-se observar que o *Oracle SQL Scratchpad* é uma ferramenta mais completa. Além disto, esta ferramenta é de fácil utilização, não necessitando de informações adicionais para o seu uso. Por estes motivos, decidiu-se utilizá-la na simulação cujos resultados são apresentados neste capítulo.

Quanto aos dados da simulação foi utilizada uma base de dados pública [TPC05] da *Transaction Processing Performance Council* (TPC). Trata-se de uma base de dados referente a um seguimento de negócio particular que gerencia, vende e distribui produtos e serviços. Na figura 4.1, pode-se observar o modelo conceitual desta base, no qual os valores numéricos dados abaixo do nome de cada tipo de entidade indicam a quantidade de registros existentes em cada uma delas, respectivamente. Tais valores são determinados em função do número de *Warehouses* (W), que para o nosso caso, foi gerado apenas um *Warehouse* (W = 1).

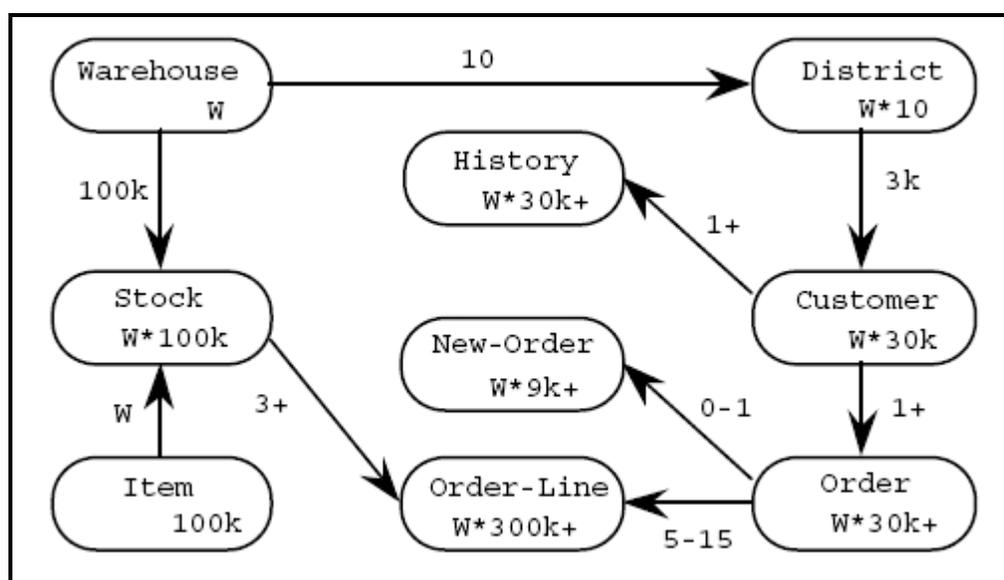


Figura 4.1 - Modelo conceitual da base de dados [TPC05].

Após escolher a base de dados para a simulação, é necessário escolher o tipo de estratégia de otimização utilizada pelo SGBD para a escolha entre os diversos planos de consulta. Neste caso, foi decidido usar a estratégia baseada em custos [Niemec03] por levar em consideração as estatísticas sobre o banco de dados. Para indicar esta escolha ao *Oracle 9i*, o parâmetro de inicialização *OPTIMIZER\_MODE* foi modificado para receber o valor *CHOOSE*.

Para utilizar esta estratégia de otimização, foi necessário gerar estatísticas sobre o banco de dados. Assim, fizemos uso do comando *ANALYZE* que permite gerar estatísticas sobre tabelas, índices e colunas. O quadro 4.2 mostra como atualizamos as estatísticas sobre

as tabelas criadas no esquema físico.

Outras estatísticas foram geradas durante a simulação de acordo com a necessidade, por exemplo, devido à criação de índices ou necessidade do armazenamento de informações sobre determinados campos de uma tabela. O comando *ANALIZE* foi usado periodicamente a fim de manter as estatísticas atualizadas.

**Quadro 4.2 – Utilizando o comando *ANALIZE* para atualizar estatísticas.**

```
ANALYZE TABLE warehouse DELETE STATISTICS;  
ANALYZE TABLE district DELETE STATISTICS;  
ANALYZE TABLE customer DELETE STATISTICS;  
ANALYZE TABLE history DELETE STATISTICS;  
ANALYZE TABLE new_order DELETE STATISTICS;  
ANALYZE TABLE orders DELETE STATISTICS;  
ANALYZE TABLE order_line DELETE STATISTICS;  
ANALYZE TABLE item DELETE STATISTICS;  
ANALYZE TABLE stock DELETE STATISTICS;  
ANALYZE TABLE warehouse COMPUTE STATISTICS;  
ANALYZE TABLE district COMPUTE STATISTICS;  
ANALYZE TABLE customer COMPUTE STATISTICS;  
ANALYZE TABLE history COMPUTE STATISTICS;  
ANALYZE TABLE new_order COMPUTE STATISTICS;  
ANALYZE TABLE orders COMPUTE STATISTICS;  
ANALYZE TABLE order_line COMPUTE STATISTICS;  
ANALYZE TABLE item COMPUTE STATISTICS;  
ANALYZE TABLE stock COMPUTE STATISTICS;
```

O experimento foi realizado gerando planos de consulta para os comandos submetidos ao sistema de banco de dados, bem como os tempos de execução para os processamentos dos mesmos. A seguir, são listados alguns casos, vistos no capítulo anterior, que foram simulados durante o desenvolvimento deste trabalho e cujos resultados são mostrados na seção 4.2:

- Índices seletivos x índices não-seletivos;
- Tipos de campos;
- Atributo *CACHE*;
- Eliminação de ordenação;
- Junções *NESTED LOOPS* x Junções *SORT-MERGE*;

- Supressão de índices;
- Cláusulas e restrições duplicadas;
- *WHERE* x *HAVING*;
- Uso de *CLUSTERS*; e
- Normalizações x Desnormalizações.

Os princípios relativos às partições e ao paralelismo não foram abordados por causa da restrição de hardware de existência de apenas uma única CPU e um único HD. A simulação do uso de distribuição também não pôde ser efetuada devida a disponibilidade de um único computador. Os *hints* foram utilizados em alguns dos casos citados anteriormente para indicar o uso de um caminho específico.

Para a realização do experimento, as únicas ferramentas em execução no SO foram o SGBD *Oracle 9i* e o *Oracle SQL Scratchpatch*. Esta precaução foi utilizada para que não existissem concorrência com outras programas, de modo que o resultado da simulação não fosse influenciado.

## 4.2. Resultados dos Experimentos

Nesta seção, serão apresentados resultados dos experimentos cujos projetos foram descritos na seção anterior. Em alguns casos, foram feitas alterações no esquema físico, como desnormalizações, mudanças de tipos e criação de índices. Além disto, foi necessário a geração de novas estatísticas, para que estas sejam atualizadas de acordo com as alterações efetuadas sobre o banco de dados. Porém, tais atualizações não serão exibidas nesta seção, devido ao número elevado de repetições, e porque elas foram feitas de forma análoga ao procedimento mostrado no quadro 4.2.

Além disso, esta seção exhibe os resultados providos pela realização dos experimentos. Seguindo uma estratégia baseada em custo, são exibidos os planos das consultas executadas, a partir das estatísticas geradas sobre a análise dos dados. Em seguida, são exibidos o custo total de cada plano e o tempo de execução da cada consulta.

### 4.2.1. Índices seletivos x índices não-seletivos

No quadro 4.3, é exibida uma consulta feita sobre a tabela *customer* a qual é restringida pelo campo *c\_state* que é uma coluna com bastantes valores distintos. Observe que não existem índices sobre este campo. Após ter sido computado o tempo de execução desta consulta, um índice sobre o campo *c\_state* da tabela *customer* foi criado (ver quadro 4.4). Este índice é bastante seletivo, pois é bastante grande o número de valores distintos deste campo. Em seguida, a consulta apresentada no quadro 4.3 foi executada novamente, para identificar mudanças no desempenho da mesma. Tais mudanças podem ser verificadas nos quadros 4.5 e 4.6.

**Quadro 4.3 – Consulta restringida pelo campo *c\_state*.**

```
SELECT *
FROM customer
WHERE c_state LIKE 'MA';
```

**Quadro 4.4 – Criando índice sobre o campo *c\_state* da tabela *customer*.**

```
CREATE INDEX indice01 ON customer(c_state);
```

**Quadro 4.5 – Plano de Consulta do comando do quadro 4.3, sem índice.**

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
2	SELECT STATEMENT	292	9	5,854
1	CUSTOMER TABLE ACCESS [FULL]	292	9	5,854
<i>Tempo de execução:</i>		<i>1,86 s</i>	<i>Custo Total de E/S:</i>	<i>292</i>

Quadro 4.6 – Plano de Consulta do comando do quadro 4.3, com índice.

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
3	SELECT STATEMENT	10	9	5,854
2	CUSTOMER TABLE ACCESS [BY INDEX ROWID]	10	9	5,854
1	SYS.INDICE01 INDEX [RANGE SCAN]	1	9	--
<b>Tempo de execução:</b>		<b>0,063 s</b>	<b>Custo Total de E/S:</b>	<b>10</b>

No quadro 4.7, uma consulta sobre a mesma tabela, porém sobre o campo *c\_credit* que possui apenas dois valores distintos, é mostrada. De forma semelhante ao experimento anterior, tal consulta foi inicialmente executada sem a existência de índices sobre o referido campo, e em seguida, ela foi processada novamente após a criação do índice sobre o campo *c\_credit* da tabela *customer* (ver quadro 4.8). Este índice não é seletivo, pois existem apenas dois valores distintos existentes nesta coluna. Pode-se ver nos quadros 4.9 e 4.10, que ambas consultas possuem o mesmo custo e o mesmo plano de execução.

Quadro 4.7 – Consulta restringida pelo campo *c\_credit*.

```
SELECT *
FROM customer
WHERE c_credit LIKE 'GC';
```

Quadro 4.8 – Criando índice sobre o campo *c\_credit* da tabela *customer*.

```
CREATE INDEX indice02 ON customer(c_credit);
```

Quadro 4.9 – Plano de Consulta do comando do quadro 4.7, sem índice.

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
2	SELECT STATEMENT	292	26.796	17.427,867
1	CUSTOMER TABLE ACCESS [FULL]	292	26.796	17.427,867
<b>Tempo de execução:</b>		<b>0,172 s</b>	<b>Custo Total de E/S:</b>	<b>292</b>

Quadro 4.10 – Plano de Consulta do comando do quadro 4.7, com índice.

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
2	SELECT STATEMENT	292	26.796	17.427,867
1	CUSTOMER TABLE ACCESS [FULL]	292	26.796	17.427,867
<b>Tempo de execução:</b>		<b>0,218 s</b>	<b>Custo Total de E/S: 292</b>	

#### 4.2.2. Tipos de campos

Uma forma de obter uma varredura completa sobre a tabela *item* é exibida no quadro 4.11. Um dos campos desta relação é o *i\_name*, que foi alterado para simular a necessidade de uma escolha adequada sobre os tipos de dados.

Quadro 4.11 – Consultando a tabela *item*.

```
SELECT *
FROM item;
```

Foi modificado o campo da tabela *item* que possuía como tipo, um texto de tamanho variável com tamanho máximo igual a vinte e quatro. O quadro 4.12 mostra como o tipo deste campo foi alterado para receber um tamanho máximo igual a duzentos e cinquenta e cinco. Então, foi realizada uma nova varredura completa sobre a tabela para verificar o efeito desta modificação, que é exibido nos quadros 4.13 e 4.14.

Quadro 4.12 – Alterando o tipo de dado do campo *i\_name* da tabela *item*.

```
ALTER TABLE item
MODIFY (i_name VARCHAR2(255));
```

Quadro 4.13 – Plano de Consulta do comando do quadro 4.11, antes da alteração do campo *i\_name*.

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
2	SELECT STATEMENT	128	100.000	8.203,125
1	ITEM TABLE ACCESS [FULL]	128	100.000	8.203,125
<b>Tempo de execução:</b>		<b>1,313 s</b>	<b>Custo Total de E/S: 128</b>	

**Quadro 4.14 – Plano de Consulta do comando do quadro 4.11, após a alteração do campo *i\_name*.**

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
2	SELECT STATEMENT	473	100.000	30.761,719
1	ITEM TABLE ACCESS [FULL]	473	100.000	30.761,719
<b>Tempo de execução:</b>		<b>0,297 s</b>	<b>Custo Total de E/S:</b>	<b>473</b>

### 4.2.3. Atributo *CACHE*

Para simular a mudança de desempenho, com o uso do atributo *CACHE*, fez-se inicialmente, uma varredura completa na tabela *orders* utilizando a consulta do quadro 4.15.

**Quadro 4.15 – Consultando a tabela *orders*.**

```
SELECT *
FROM orders;
```

Em seguida, a tabela *new\_order* foi modificada, para que os dados da mesma fossem armazenados em *CACHE*. O quadro 4.16 exibe o comando DDL que foi usado para alteração desta tabela. Então, a consulta do quadro 4.15 foi executada mais duas vezes, depois que a relação *new\_order* foi alterada, e resultados obtidos mostraram que o plano de consulta permanecia inalterado. Porém, o tempo de execução diminuía cada vez mais, como se pode ver nos quadros 4.17, 4.18 e 4.19 .

**Quadro 4.16 – Alterando a tabela *orders* para ser armazenada em *CACHE*.**

```
ALTER TABLE orders CACHE;
```

Quadro 4.17 – Plano de Consulta do comando do quadro 4.15, sem uso do atributo CACHE.

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
2	SELECT STATEMENT	25	29.837	1.136,37
1	ORDERS TABLE ACCESS [FULL]	25	29.837	1.136,37
<i>Tempo de execução:</i>		<i>0,188 s</i>	<i>Custo Total de E/S:</i>	<i>25</i>

Quadro 4.18 – Plano de Consulta do comando do quadro 4.15, após primeira execução usando o atributo CACHE.

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
2	SELECT STATEMENT	25	29.837	1.136,37
1	ORDERS TABLE ACCESS [FULL]	25	29.837	1.136,37
<i>Tempo de execução:</i>		<i>0,079 s</i>	<i>Custo Total de E/S:</i>	<i>25</i>

Quadro 4.19 – Plano de Consulta do comando do quadro 4.15, após segunda execução usando o atributo CACHE.

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
2	SELECT STATEMENT	25	29.837	1.136,37
1	ORDERS TABLE ACCESS [FULL]	25	29.837	1.136,37
<i>Tempo de execução:</i>		<i>0,032 s</i>	<i>Custo Total de E/S:</i>	<i>25</i>

#### 4.2.4. Eliminação de Ordenação

O quadro 4.20 exibe uma consulta sobre a relação *customer* utilizando as cláusulas *GROUP BY* e *ORDER BY*. Esta consulta retorna o número de clientes para cada tipo de crédito, ordenado pelo tipo de crédito.

Quadro 4.20 – Consultando a tabela *customer*, utilizando as cláusulas *GROUP BY* e *ORDER BY*.

```
SELECT c_credit, count(*)
FROM customer
GROUP BY c_credit
ORDER BY c_credit;
```

A consulta, apresentada no quadro 4.21, gera o mesmo resultado do comando

apresentado no quadro 4.20. Isto se dá pelo fato da cláusula *GROUP BY* utilizar uma ordenação implícita para realizar o agrupamento dos dados. Resultados de comprovação deste fato são mostrados nos quadros 4.22 e 4.23.

**Quadro 4.21 – Consultando a tabela customer, utilizando a cláusulas GROUP BY.**

```
SELECT c_credit, count(*)
FROM customer
GROUP BY c_credit;
```

**Quadro 4.22 – Plano de Consulta do comando do quadro 5.20.**

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
4	SELECT STATEMENT	365	2	0,004
3	SORT [ORDER BY]	365	2	0,004
2	SORT [GROUP BY]	365	2	0,004
1	CUSTOMER TABLE ACCESS [FULL]	292	29.837	58,275
<b>Tempo de execução:</b>		<b>1,515 s</b>	<b>Custo Total de E/S:</b>	<b>365</b>

**Quadro 4.23 – Plano de Consulta do comando do quadro 5.21.**

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
3	SELECT STATEMENT	329	2	0,004
2	SORT [GROUP BY]	329	2	0,004
1	CUSTOMER TABLE ACCESS [FULL]	292	29.837	58,275
<b>Tempo de execução:</b>		<b>1,125 s</b>	<b>Custo Total de E/S:</b>	<b>329</b>

#### 4.2.5. Junções NESTED LOOPS x Junções SORT-MERGE

O quadro 4.24 mostra como pode ser feita uma junção entre as tabelas *district* e *customer*, no qual é usado o *hint USE\_NL* para indicar ao SGBD, a utilização do tipo de junção *NESTED LOOPS*. Observe que também foi usado o *hint ORDERED*, para que o *Oracle 9i* não escolha uma junção do tipo *HASH*. A junção *HASH* poderia ser a mais adequada nesta situação, pela relação *district* possuir poucos registros.

**Quadro 4.24 – Junção NESTED LOOPS entre district e customer.**

```
SELECT /*+ USE_NL(district) ORDERED */ d_name, c_first,
c_last, c_phone
FROM customer, district
WHERE d_id = c_d_id
      AND d_w_id = c_w_id;
```

Na seqüência dos experimentos, a mesma junção foi realizada, só que modificando o tipo de junção para *MERGE-SORT*. Para que a junção *MERGE-SORT* seja utilizada, usamos os *hints* *USE\_MERGE* e *ORDERED* como pode ser visto no quadro 4.25. A diferença entre os custos das junções efetuadas pode ser vista nos quadros 4.26 e 4.27.

**Quadro 4.25 – Junção MERGE-SORT entre district e customer.**

```
SELECT /*+ USE_MERGE(district) ORDERED */ d_name, c_first,
c_last, c_phone
FROM customer, district
WHERE d_id = c_d_id
      AND d_w_id = c_w_id;
```

**Quadro 4.26 – Plano de Consulta do comando do quadro 4.24.**

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
5	SELECT STATEMENT	30.129	29.837	1.923,088
4	NESTED LOOPS	30.129	29.837	1.923,088
1	CUSTOMER TABLE ACCESS [FULL]	292	29.837	1.515,16
3	DISTRICT TABLE ACCESS [BY INDEX ROWID]	1	1	0,014
2	SYS_C003009 INDEX [UNIQUE SCAN]	--	1	--
<b>Tempo de execução:</b>		<b>0,031 s</b>	<b>Custo Total de E/S:</b>	<b>30.129</b>

Quadro 4.27 – Plano de Consulta do comando do quadro 4.25.

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
6	SELECT STATEMENT	607	29.837	1.923,088
5	MERGE JOIN	607	29.837	1.923,088
2	SORT [JOIN]	604	29.837	1.515,16
1	CUSTOMER TABLE ACCESS [FULL]	292	29.837	1.515,16
4	SORT [JOIN]	4	10	0,137
3	DISTRICT TABLE ACCESS [FULL]	2	10	0,137
<b>Tempo de execução:</b>		<b>7,937 s</b>	<b>Custo Total de E/S:</b>	<b>607</b>

#### 4.2.6. Supressão de Índices

A consulta, exibida no quadro 4.28, é restringida por um campo indexado. Vimos anteriormente que este índice é bastante seletivo, possuindo portanto, altas chances de ser utilizado quando possível como pode-se ver no quadro 4.31..

Quadro 4.28 – Consulta restringida por campo indexado.

```
SELECT c_first, c_middle, c_last, c_phone
FROM customer
WHERE c_state LIKE 'PE'
      OR c_state LIKE 'MA';
```

Por outro lado, a consulta do quadro 4.29, utiliza o predicado *IS NULL* sobre o campo *c\_state* que está indexado. Foi vista a influência do uso deste predicado sobre a consulta, cujo plano de consulta é exibido pelo quadro 4.32.

Quadro 4.29 – Consulta restringida por campo indexado usando IS NULL.

```
SELECT c_first, c_middle, c_last, c_phone
FROM customer
WHERE c_state LIKE 'PE'
      OR c_state IS NULL;
```

O quadro 4.30 mostra o uso do predicado *IS NOT NULL* sobre o campo *c\_state*. Observe que a cláusula *WHERE* poderia ter sido reescrita de forma que o predicado *IS NOT*

*NULL* não fosse utilizado, e mesmo assim, gerando o mesmo resultado como podemos ver no quadro 4.33.

**Quadro 4.30 – Consulta restringida por campo indexado usando IS NOT NULL.**

```
SELECT c_first, c_middle, c_last, c_phone
FROM customer
WHERE c_state LIKE 'PE'
      AND c_state IS NOT NULL;
```

**Quadro 4.31– Plano de Consulta do comando do quadro 4.28.**

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
4	SELECT STATEMENT	20	18	0,914
3	INLIST ITERATOR			
2	CUSTOMER TABLE ACCESS [BY INDEX ROWID]	20	18	0,914
1	INDICE01 INDEX [RANGE SCAN]	2	18	--
<b>Tempo de execução:</b>		<b>0,094 s</b>	<b>Custo Total de E/S:</b>	<b>20</b>

**Quadro 4.32 – Plano de Consulta do comando do quadro 4.29.**

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
2	SELECT STATEMENT	292	9	0,457
1	CUSTOMER TABLE ACCESS [FULL]	292	9	0,457
<b>Tempo de execução:</b>		<b>1,000 s</b>	<b>Custo Total de E/S:</b>	<b>292</b>

**Quadro 4.33 – Plano de Consulta do comando do quadro 4.30.**

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
3	SELECT STATEMENT	10	9	0,457
2	CUSTOMER TABLE ACCESS [BY INDEX ROWID]	10	9	0,457
1	INDICE01 INDEX [RANGE SCAN]	1	9	--
<b>Tempo de execução:</b>		<b>1,250 s</b>	<b>Custo Total de E/S:</b>	<b>20</b>

## 4.2.7. Cláusulas e Restrições Duplicadas

O uso da cláusula *DISTINCT* é mostrado no quadro 4.34, para retornar tuplas distintas contendo os campos *o\_w\_id*, *o\_d\_id* e *o\_id* da relação *orders*. Porém, estes são todos os campos que formam a chave primária desta relação. Assim, as tuplas retornadas seriam únicas mesmo sem o uso da cláusula *DISTINCT*.

**Quadro 4.34 – Consultando a chave primária da tabela *orders* usando *DISTINCT*.**

```
SELECT DISTINCT o_w_id, o_d_id, o_id
FROM orders;
```

No quadro 4.35, a consulta foi reescrita de forma que a cláusula *DISTINCT* foi eliminada. O mesmo resultado, da consulta do quadro 4.34, foi obtido em termos de tuplas retornadas. Mas em termos de desempenho, a consulta do quadro 4.35 foi mais eficiente. Tais mudanças podem ser verificadas nos quadros 4.36 e 4.37.

**Quadro 4.35 – Consultando a chave primária da tabela *orders*.**

```
SELECT o_w_id, o_d_id, o_id
FROM orders;
```

**Quadro 4.36 – Plano de Consulta do comando do quadro 4.34.**

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
3	SELECT STATEMENT	76	29.837	203,964
2	SORT [UNIQUE]	76	29.837	203,964
1	SYS_C003014 INDEX [FAST FULL SCAN]	9	29.837	203,964
<b>Tempo de execução:</b>		<b>2,047 s</b>	<b>Custo Total de E/S: 76</b>	

**Quadro 4.37 – Plano de Consulta do comando do quadro 4.35.**

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
2	SELECT STATEMENT	9	29.837	203,964
1	SYS_C003014 INDEX [FAST FULL SCAN]	9	29.837	203,964

<b>Tempo de execução:</b>	<b>0,063 s</b>	<b>Custo Total de E/S:</b>	<b>9</b>
---------------------------	----------------	----------------------------	----------

#### 4.2.8. WHERE x HAVING

A consulta do quadro 4.38 utiliza a função de agregação *COUNT* para contar o número de registros de um certo tipo de crédito restringido pela cláusula *HAVING*.

**Quadro 4.38 – Utilizando a cláusula HAVING.**

```
SELECT c_credit, COUNT(*)
FROM customer
HAVING c_credit = 'GC'
GROUP BY c_credit;
```

A consulta do quadro 4.39 retorna o mesmo conjunto de resultados da consulta do quadro 4.38. Contudo, esta última utiliza a cláusula *WHERE* no lugar de *HAVING*. Resultados, vistos nos quadros 4.40 e 4.41, indicam que a consulta do quadro 4.39 é mais eficiente.

**Quadro 4.39 – Utilizando a cláusula WHERE.**

```
SELECT c_credit, COUNT(*)
FROM customer
WHERE c_credit = 'GC'
GROUP BY c_credit;
```

**Quadro 4.40 – Plano de Consulta do comando do quadro 5.38.**

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
4	SELECT STATEMENT	329	1	0,002
3	FILTER			
2	SORT [GROUP BY]	329	1	0,002
1	CUSTOMER TABLE ACCESS [FULL]	292	29.837	58,275
<b>Tempo de execução:</b>		<b>1,578 s</b>	<b>Custo Total de E/S:</b>	<b>329</b>

Quadro 4.41 – Plano de Consulta do comando do quadro 5.39.

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
3	SELECT STATEMENT	292	26.796	52,336
2	SORT [GROUP BY NOSORT]	292	26.796	52,336
1	CUSTOMER TABLE ACCESS [FULL]	292	26.796	52,336
<b>Tempo de execução:</b>		<b>1,110 s</b>	<b>Custo Total de E/S:</b>	<b>292</b>

#### 4.2.9. Uso de clusters

Inicialmente, foi criado um *cluster* segundo mostra o quadro 4.42. Então, foram criadas duas tabelas idênticas às tabelas *customer* e *history*, chamadas respectivamente de *customer2* e *history2*. Também foi criado um relacionamento entre as tabelas *customer2* e *history2*, semelhante ao relacionamento existente entre as tabelas *customer* e *history*.

Quadro 4.42 – Criando o cluster *customer\_history*.

```
CREATE CLUSTER customer_history (
  C_W_ID NUMBER(6),
  C_D_ID NUMBER(6),
  C_ID NUMBER(6)
);
```

O quadro 4.43, mostra a criação de um índice sobre o *cluster customer\_history*. Em seguida, as tabelas *customer2* e *history2* foram populadas, respectivamente, com os mesmos dados das relações *customer* e *history*.

Quadro 4.43 – Criando o cluster *customer\_history*.

```
CREATE INDEX indice03 ON CLUSTER customer_history;
```

Uma consulta realizando junções entre as tabelas *customer* e *history* foi processada, conforme mostra o quadro 4.44. Em seguida, a mesma consulta foi realizada para as tabelas *customer2* e *history2* (ver quadro 4.45). Podemos visualizar o custo destas consultas nos

quadro 4.46 e 4.47

**Quadro 4.44 – Consulta envolvendo uma junção entre customer e history.**

```
SELECT h_date
FROM history, customer
WHERE c_w_id = h_c_w_id
      AND c_d_id = h_c_d_id
      AND c_id = h_c_id
      AND h_c_id = 1;
```

**Quadro 4.45 – Consulta envolvendo uma junção entre customer2 e history2.**

```
SELECT h_date
FROM customer2, history2
WHERE c_w_id = h_c_w_id
      AND c_d_id = h_c_d_id
      AND c_id = h_c_id
      AND h_c_id = 1;
```

**Quadro 4.46 – Plano de Consulta do comando do quadro 5.44.**

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
4	SELECT STATEMENT	31	10	0,361
3	NESTED LOOPS	31	10	0,361
1	HISTORY TABLE ACCESS [FULL]	31	10	0,293
2	SYS_C003011 INDEX [UNIQUE SCAN]	--	1	0,007
<b>Tempo de execução:</b>		<b>0,469 s</b>	<b>Custo Total de E/S:</b>	<b>31</b>

**Quadro 4.47 – Plano de Consulta do comando do quadro 5.45.**

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
5	SELECT STATEMENT	19	10	0,361
4	NESTED LOOPS	19	10	0,361
1	SYS_C003027 INDEX [FAST FULL SCAN]	9	10	0,068
3	HISTORY2 TABLE ACCESS [CLUSTER]	1	1	0,029
2	INDICE03 INDEX [UNIQUE SCAN]	--	2.984	--
<b>Tempo de execução:</b>		<b>0,656 s</b>	<b>Custo Total de E/S:</b>	<b>19</b>

#### 4.2.10. Normalização x Desnormalização

Para este caso, foi criada a tabela *cust\_hist* que resulta da desnormalização das tabelas *customer* e *history*. A consulta do quadro 4.48 exibe dados provenientes destas duas tabelas.

**Quadro 4.48 – Consultando dados das tabelas *customer* e *history*.**

```
SELECT *
FROM customer, history
WHERE c_w_id = h_c_w_id
      AND c_d_id = h_c_d_id
      AND c_id = h_c_id;
```

No quadro 4.49, pode-se observar que todos dados necessários estão presentes na tabela desnormalizada *cust\_hist*. Assim, não existe a necessidade de junção com outra tabela, como pode ser observado nos quadros 4.52 e 4.53.

**Quadro 4.49 – Consultando dados da tabela *cust\_hist*.**

```
SELECT *
FROM cust_hist;
```

O quadro 4.50 exibe uma consulta na qual as informações necessárias estão todas mantidas na tabela *customer*. A consulta do quadro 4.51 obtém o mesmo resultado em termos de tuplas retornadas, porém a consulta é realizada sobre a tabela desnormalizada *cust\_hist*. Na qual, a diferença entre os custos desta consultas pode ser vista nos quadros 4.54 e 4.55.

**Quadro 4.50 – Consultando a tabela *customer*.**

```
SELECT c_first, c_middle, c_last, c_city, c_state
FROM customer;
```

Quadro 4.51 – Consultando dados existentes na customer através de cust\_hist.

```
SELECT c_first, c_middle, c_last, c_city, c_state
FROM cust_hist;
```

Quadro 4.52 – Plano de Consulta do comando do quadro 5.48.

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
4	SELECT STATEMENT	1.229	29.837	21.153,967
3	HASH JOIN	1.229	29.837	21.153,967
1	HISTORY TABLE ACCESS [FULL]	31	29.837	1.748,262
2	CUSTOMER TABLE ACCESS [FULL]	292	29.837	19.405,705
<b>Tempo de execução:</b>		<b>0,687 s</b>	<b>Custo Total de E/S:</b>	<b>1.229</b>

Quadro 4.53 – Plano de Consulta do comando do quadro 5.49.

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
2	SELECT STATEMENT	322	29.837	20.950,003
1	CUST_HIST TABLE ACCESS [FULL]	322	29.837	20.950,003
<b>Tempo de execução:</b>		<b>2,031 s</b>	<b>Custo Total de E/S:</b>	<b>322</b>

Quadro 4.54 – Plano de Consulta do comando do quadro 5.50.

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
2	SELECT STATEMENT	292	29.837	1.631,711
1	CUSTOMER TABLE ACCESS [FULL]	292	29.837	1.631,711
<b>Tempo de execução:</b>		<b>0,469 s</b>	<b>Custo Total de E/S:</b>	<b>292</b>

Quadro 4.55 – Plano de Consulta do comando do quadro 5.51.

<i>Ordem</i>	<i>Etapa</i>	<i>Custo E/S</i>	<i>Regs Retornados</i>	<i>KBytes Retornados</i>
2	SELECT STATEMENT	322	29.837	1.631,711
1	CUST_HIST TABLE ACCESS [FULL]	322	29.837	1.631,711
<b>Tempo de execução:</b>		<b>1,109 s</b>	<b>Custo Total de E/S:</b>	<b>322</b>

## 4.3. Análise dos Resultados

Nesta seção, realizamos uma análise a partir dos resultados obtidos com a realização dos experimentos.

### 4.3.1. Índices seletivos x índices não-seletivos

Pudemos observar, a partir dos quadros 4.5 e 4.6, que a criação do índice sobre o campo *c\_state* da relação *customer* diminuiu o custo da consulta, mostrada no quadro 4.3, para menos de 4% do custo sem a utilização de índice. Como a coluna *c\_state* possui vários valores distintos, o índice criado passa a ser bastante seletivo. Desta forma, seu uso é recomendado para melhorar o desempenho como pode-se ver no plano de consulta exibido no quadro 4.6.

O mesmo experimento foi realizado para o campo *c\_credit* que possui apenas dois valores diferentes. Sendo assim, o índice criado sobre este campo é pouco seletivo, e como pôde-se observar nos quadros 4.9 e 4.10, o índice não foi utilizado após a sua criação. Ambos planos de consulta, apresentados pelos quadros 4.9 e 4.10, possuem o mesmo custo e fazem uma varredura completa na relação *customer*.

### 4.3.2. Tipos de campos

A consulta do quadro 4.11 foi executada duas vezes, realizando duas varreduras completas sobre a relação *item*. Antes da segunda execução da consulta, o tipo do campo *i\_name* foi modificado de forma inadequada. Pudemos ver, pelos quadros 4.13 e 4.14, que o custo praticamente triplicou, bem como a quantidade de *bytes* retornados pela consulta, após a alteração do tipo do campo.

### 4.3.3. Atributo CACHE

O quadro 4.15 apresenta uma consulta, que executa uma varredura completa na relação *orders*. Foram realizadas três execuções para esta mesma consulta. Após a primeira execução, a tabela *orders* foi alterada para que a mesma fosse armazenada na *CACHE*. Pode-se observar pelos quadros 4.17, 4.18 e 4.19 que o tempo de execução da consulta diminuiu a cada execução seguinte. Isto ocorre, porque os dados desta relação passaram a ser armazenados em memória primária.

### 4.3.4. Eliminação de Ordenação

A consulta do quadro 4.20 possui as cláusulas *GROUP BY* e *ORDER BY*, executando operações de ordenação em duplicidade. No quadro 4.21, a cláusula *ORDER BY* foi removida, mas o mesmo conjunto de resultados foi obtido a partir da execução da consulta alterada. Observando os quadros 4.22 e 4.23, pode-se ver que houve um ganho de performance devido à eliminação da operação de ordenação.

### 4.3.5. Junções NESTED LOOPS x Junções SORT-MERGE

Os quadros 4.24 e 4.25 exibem duas consultas sobre os mesmos dados, porém utilizando *hints* de junções diferentes. A primeira consulta realiza uma junção *NESTED LOOPS* sobre as relações *customer* e *district*. Esta consulta apresenta um custo bastante alto, como mostra o quadro 4.26, pois a relação controladora apresenta uma grande quantidade de registros. Pode-se ver também, através deste plano de consulta, que o SGBD criou o índice *SYS\_C003009* implicitamente na criação da tabela *district*.

O quadro 4.27, que exhibe o plano da consulta do quadro 4.25, mostra uma junção

*SORT-MERGE* com custo bem inferior ao da primeira consulta. Pode-se observar que o tempo de execução foi superior a sete segundos. Este fato ocorre por uma das operações de ordenação, efetuadas nesta junção, ter que ordenar cerca de trinta mil registros ou 1,5 milhões de bytes, necessitando assim, de um alto valor para o parâmetro de inicialização *SORT\_AREA\_SIZE*.

#### **4.3.6. Supressão de Índices**

Podemos observar que a consulta, exibida no quadro 4.28, é restringida por dois valores, para o campo indexado *c\_state* da relação *customer*. O plano desta consulta exibido no quadro 4.31, mostra que o índice sobre *customer* é utilizado. Na consulta do quadro 4.29, é utilizada a expressão *IS NULL* fazendo com que o índice fosse suprimido, como é mostrado no quadro 4.32.

A consulta do quadro 4.30 usa a expressão *IS NOT NULL*, e é fácil verificar que a mesma poderia ter sido reescrita de forma a eliminar esta expressão, e mesmo assim, gerando o mesmo resultado. Porém, o SGBD consegue perceber este fato e utiliza o índice sobre o campo *c\_state*, como pode ser visto no quadro 4.33.

#### **4.3.7. Cláusulas e Restrições Duplicadas**

O quadro 4.34 mostra o uso da cláusula *DISTINCT* sobre todos os campos que formam a chave primária da relação *orders*. Esta cláusula retorna todas as tuplas distintas na consulta, realizando uma operação de ordenação implicitamente, que é exibido pelo quadro 4.36. Todavia, o uso desta cláusula pode ser evitado neste caso, como podemos ver no quadro 4.35, pois conjunto de valores dos campos de uma chave primária são distintos por definição. Por isso, o quadro 4.37 exhibe um custo bem inferior ao apresentado pelo quadro 4.36.

### 4.3.8. WHERE x HAVING

A consulta, exibida no quadro 4.38, realiza um agrupamento sobre a relação *customer*. Esta consulta está restringida pela utilização da cláusula *HAVING*, e pode-se ver pelo quadro 4.40, que esta seleção é feita após a operação de ordenação implícita, efetuada pelo *GROUP BY*. O quadro 4.39 mostra uma consulta na qual esta restrição é feita utilizando a cláusula *WHERE*. Observe que pelo quadro 4.41, a operação de seleção é feita antes da ordenação, de forma que apenas os dados necessários são ordenados, diminuindo assim, o custo da consulta, após sua reescrita.

### 4.3.9. Uso de clusters

Este experimento fez uso de uma consulta que realiza a junção sobre as tabelas *customer* e *history*, no quadro 4.44. O quadro 4.45 apresenta uma junção sobre as tabelas *customer2* e *history2*. Estas tabelas possuem, respectivamente, os mesmos dados e esquema lógico que as relações *customer* e *history*. Porém, as tabelas *customer2* e *history2* estão presentes em um mesmo *cluster*. Pelos quadros 4.46 e 4.47, pode-se verificar que a segunda consulta possui um ganho de performance sobre a primeira devido a utilização da estrutura adicional.

### 4.3.10. Normalização x Desnormalização

O quadro 4.48 exhibe uma consulta sobre dados das relações *customer* e *history*. A mesma consulta foi realizada sobre a relação desnormalizada *cust\_hist* como é mostrado no quadro 4.49. Pode-se ver pelos quadros 4.52 e 4.53, que a segunda consulta é mais eficiente que a primeira. Isto ocorreu porque na primeira consulta, foi necessário efetuar uma

operação de junção, que é uma operação relativamente cara, para a obtenção dos dados.

A consulta, do quadro 4.50, exibe dados referentes a relação *customer* e exibiu um desempenho melhor que a mesma consulta para a relação desnormalizada *cust\_hist*, no quadro 4.51. O custo adicional sobre a relação desnormalizada, mostrado nos quadros 4.54 e 4.55, deveu-se ao fato de ser necessário varrer uma maior quantidade de dados.

## 4.4. Conclusão

Neste capítulo, projetamos alguns experimentos e exibimos os resultados da simulação dos mesmos. A partir da simulação destes experimentos, obteve-se dados necessários para realização de uma análise. Para a simulação, foi utilizada a ferramenta *Oracle SQL Scratchpad* presente no *Oracle Enterprise Manager*. Esta ferramenta faz parte do pacote de instalação do SGBD *Oracle 9i* e mostrou-se ser adequada para esta simulação, por apresentar certas funcionalidades como exibir planos de consulta de modo prático e tempos de execução para consultas.

Nos experimentos, também foi utilizado o comando *ANALYSE* para gerar estatísticas sobre uma base de dados pública [TPC05]. Uma das estatísticas exibidas foi o tempo de execução da consulta, utilizado apenas para o caso de armazenamento em *CACHE*, por apresentar diversos valores para cada execução de uma mesma consulta e depender dos dados presentes em memória principal.

Finalmente, a partir da análise dos resultados gerados pela realização do experimento, pôde-se comprovar os princípios de sintonia de esquemas e consultas investigados neste trabalho. No quadro 4.56, é exibido um resumo da análise dos resultados obtidos na simulação.

Quadro 4.56 – Resumo da simulação dos princípios.

	<i>Princípios</i>	<i>Resultados</i>
<b>P1</b>	Índice seletivo	Uso de índice recomendado
	Índice não seletivo	Índice desnecessário
<b>P2</b>	Tipo de campo adequado	Recomendado para que haja consultas eficientes
	Tipo de campo inadequado	Podem gerar problemas de desempenho
<b>P3</b>	Tabelas em <i>CACHE</i>	Para tabelas pequenas que necessitam de eficiência
	Tabelas em disco	Para tabelas cujo desempenho não é crítico
<b>P4</b>	Ordenações duplicadas	Uso desnecessário
	Eliminação de ordenações	Efetuar no máximo uma ordenação
<b>P5</b>	Junção <i>NESTED LOOPS</i>	Quando a tabela externa é pequena, e a interna é acessada por índice
	Junção <i>SORT-MERGE</i>	Quando não existem índices e as tabelas são pequenas
<b>P6</b>	Uso de <i>IS NULL</i>	Deve ser evitado
<b>P7</b>	Restrições duplicadas	Devem ser evitadas
<b>P8</b>	Uso de <i>WHERE</i>	Recomendado sempre que possível
	Uso de <i>HAVING</i>	Deve ser evitado sempre que possível
<b>P9</b>	Tabelas acessadas juntas	Recomendado o uso de <i>cluster</i>
	Tabelas acessadas separadas	Não recomendado o uso de <i>cluster</i>
<b>P10</b>	Tabelas acessadas juntas	Desnormalização é recomendada
	Tabelas acessadas separadas	Normalização é recomendada

## 5. Conclusão

Este capítulo conclui a apresentação deste trabalho. Na seção 5.1 são citadas as principais contribuições do trabalho realizado. Finalmente, a seção 5.2 propõe um conjunto de trabalhos futuros para serem realizados.

### 5.1. Principais Contribuições

Neste trabalho, observou-se que a sintonia de banco de dados pode ser efetuada de várias formas. Porém, mudanças feitas sobre o hardware podem ser relativamente caras. Desta forma, é preferível que estes ajustes sejam feitos no lado do software. Por isso, investigamos, como uma das formas de sintonia de banco de dados, o ajuste de esquemas e consultas pelos ABD, desenvolvedores e especialistas, poderia ser realizada e validada.

Para estes tipos de sintonia, foi feita uma análise sobre as diferentes formas de esquemas e consultas, e suas influências sobre o desempenho de um banco de dados. Esta análise forneceu subsídios para a identificação de um conjunto de princípios. Além disso, foi observada a importância da interação entre consultas e esquemas, de modo que ambos estejam ajustados para uma melhor performance do BD.

A partir de uma necessidade de validação das idéias propostas neste trabalho, foi realizada uma simulação usando o *Oracle 9i* e o *Oracle SQL Scratchpad* do *Oracle Enterprise Manager*. Para esta simulação, foi utilizada uma base de dados exemplo. Através de uma análise sobre os resultados de uma simulação, pôde-se comprovar a eficiência de alguns princípios apresentados.

### 5.2. Trabalhos Futuros

Atualmente, os SGBD não são totalmente independentes de seus usuários para a

questão da sintonia de esquemas e consultas. A criação de ferramentas automáticas que possam auxiliar o ABD nos ajustes de esquemas, consultas e configurações do SGBD, é um trabalho de grande importância para a área de sintonia de BD. Em [Salles04] é estudada a criação de índices hipotéticos, com base no custo de criação destes índices e nos benefícios gerados por estes.

Uma área de pesquisa citada, mas não aprofundada neste trabalho, é a sintonia de Bancos de Dados Distribuídos (BDD). Os bancos de dados distribuídos ainda são recentes, e por isso, há muito o que se estudar nesta área. Ajustes para obter uma maior eficiência podem incluir diversas sub-áreas como as escolhas da distribuição dos dados entre os sites, e do plano de execução para uma consulta.

Finalmente, outra sugestão é a criação de uma ferramenta que auxilie os ABD, os desenvolvedores e os especialistas, a criarem consultas e esquemas mais eficientes. E que possa indicar a estes usuários de um SGBD, qual quantidade de tuplas pode ser processada com um tempo de resposta aceitável.

## Referências

[**Baylis01**] Ruth Baylis, Kathy Rich and Joyce Fee, “Oracle9i Database Administrator’s Guide”, Oracle Corporation, 2001.

[**Bultzingsloewen89**] Günter Von Bultzingsloewen, “Optimizing SQL Queries for Parallel Execution”, Sigmod Record, 1989.

[**Elmasri99**] Ramez A. Elmasri and Shamkant B. Navathe, “Fundamentals of Database Systems”, Addison-Wesley, 1999.

[**England01**] Ken England, “Microsoft SQL Server 2000 Performance Optimization and Tuning Handbook”, Digital Press, 2001.

[**Fernandes02**] Lúcia Fernandes, “Oracle 9i para Desenvolvedores - Oracle Developer 6i - Curso Completo”, Axcel Books, 2002.

[**Ganski87**] Richard Ganski and Harry K. T. Wong, “Optimization of nested SQL queries revisited”, in Umeshwar Dayal and Irving L. Traiger, editors, Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, San Francisco, May 27-29, 1987, 23-33, ACM Press, 1987.

[**Garcia00**] E. Garcia-Molina, J. D. Ullman & J. Widon, “Database System Implementation”, Prentice-Hall, 2000.

[**Joshi01**] Ceptsar A. Galindo-Legaria and Milind Joshi, “Orthogonal optimization of subqueries and aggregation”, in Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, Santa Barbara, Calif., May 21-24, 2001, 571-581, ACM Press, 2001.

**[Kemme05]** Bettina Kemme, “Query Optimization”, School of Computer Science McGill University, 2005.

**[Kim82]** Won Kim, “On optimizing an SQL-like nested query”, TODS, 7(3):443-469, 1982.

**[Korth01]** Abraham Silberschatz, Henry F. Korth, and S. Sudarshan, “Database Systems Concepts”, McGraw-Hill Higher Education, 2001.

**[Lehman86]** Tobin J. Lehman and Michael J. Carey, “A study of index structures for main memory database management systems”, in VLDB’86 Twelfth International Conference on Very Large Data Bases, August 25-28, Kyoto, Japan, 294-303, Morgan Kaufmann, 1986.

**[Lifschitz04]** Rogério Luís de Carvalho Costa e Sérgio Lifschitz, “Sintonia e Auto-Sintonia em Banco de dados”, Apresentação sobre trabalho no Congresso da Sociedade Brasileira de Computação, 2004.

**[Loney01]** Kevin Loney and Marlene Thieralt, “Oracle9i DBA Handbook”, Oracle Press, 2001.

**[Loney02]** Kevin Loney, George Koch and Experts at TUSC, “Oracle 9i: The Complete Reference”, McGraw-Hill and Osborne, 2002.

**[Lorentz01]** Diana Lorentz, “Oracle9i SQL Reference”, Oracle Corporation, 2001.

**[Manber89]** Udi Manber, “Introduction to Algorithms - A Creative Approach”, Addison-Wesley, 1989.

**[Niemec03]** Richard J. Niemec, Bradley D. Brown and Joseph C. Trezzo, “Oracle 9i – Performance Tuning”, Oracle Press, 2003.

**[Oracle01]** Oracle, “Oracle 9i, Database Performance Guide and Reference”, Oracle Press, 2001.

**[Ozsu99]** M. Tamer Özsu and Patrick Valduriez, “Principles of Distributed Database Systems”, Prentice-Hall, 1999.

**[Patterson98]** David A. Patterson and John L. Hennessy, “Computer Organization and Design – The hardware/software interface”, Morgan Kaufmann, 1998.

**[Powell04]** Gavin Powell, “Oracle High Performance Tuning for 9i and 10g”, Digital Press, 2004.

**[Salles04]** Marcos Antônio Vaz Salles, “Definição, implementação e avaliação de uma ferramenta de auto-sintonia de índices baseada em agentes”, Dissertação de Mestrado do Departamento de Informática da PUC-RJ, 2004.

**[Sasha03]** Dennis Sasha and Philippe Bonnet, “Database Tuning - Principles, Experiments, and Troubleshooting Techniques”, Morgan Kaufmann, 2003.

**[Silva03]** Aleksandra Cassiano Alves da Silva, “Análise de Formas de Otimização de Consultas”, Trabalho de Graduação, 2003.

**[TPC05]** Transaction Processing Performance Council, “TPC BENCHMARK™ C – Standard Specification Revision 5.4”, Transaction Processing Performance Council, 2005.

**[Whalen96]** Edward Whalen, “Oracle Performance Tuning and Optimization”, Sams Publishing, 1996.