Análise de Desempenho de Banco de Dados utilizando *Benchmarks* Especializados

Este exemplar corresponde a redação final da tese devidamente corrigida e defendida pelo Sr. *Rafles Frausino Pereira* e aprovada pela Comissão Julgadora.

Campinas, 19 de setembro de 1990.

Prof. Dr. *Geovane Cayres Magalhães*

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para obtenção do Título de Mestre em CIÊNCIA DA COMPUTAÇÃO.

AGRADECIMENTOS

À Cláudia por seu amor e compreensão.

Aos meus pais pelo apoio e estímulo constantes.

Ao prof. Geovane Cayres Magalhães por suas preciosas orientações e amizade.

Ao prof. Rogério Drummond pelas orientações na fase inicial deste curso de mestrado.

À Telebrás pelo apoio e compreensão, sem os quais este trabalho não chegaria ao final.

Ao CPqD/Telebrás pela disponibilidade dos vários recursos de banco de dados, comunicação de dados e impressão, necessários para a elaboração desta tese.

Finalmente, agradeço à Miguel Frauzino por sua cuidadosa revisão e sugestões sobre o texto.

Análise de Desempenho de Banco de Dados utilizando *Benchmarks* Especializados

por: Rafles Frausino Pereira
orientador: Prof. Geovane Cayres Magalhães

SUMÁRIO

Apresentamos, neste trabalho, uma metodologia de análise de desempenho de sistemas de bancos de dados intitulada *Metodologia de Benchmark Especializado* (MBE). Nesta metodologia, o usuário parametriza um *modelo de representação do sistema*, baseado em uma aplicação ou conjunto de aplicações. O modelo é estruturado em *níveis*, que representam desde as estruturas mais abstratas de um sistema de banco de dados, até as características mais físicas. Nesta representação podem ser modelados: o *ambiente de execução*, o *banco de dados* e a *carga de trabalho*. A metodologia pode ser empregada na análise de alternativas de *projeto* de banco de dados, como, também, na seleção de um sistema de gerenciamento de banco de dados. A validação da MBE foi efetuada através da implementação de um protótipo de uma ferramenta, que foi utilizada para gerar um *benchmark* sintético conhecido e um *benchmark* especializado para uma dada aplicação.

ABSTRACT

This work describes a database performance methodology called *Specialized Benchmark Methodology* (MBE). In this methodology the user sets parameters of a *layered system model* that is based on one or a group of aplications. Each layer represents an abstraction level of an application representation, from the logical structure down to the most physical aspects. The *environment*, *the database* and the *workload* can be represented in the model. The MBE can be used for selecting design alternatives as well as in the selection of database management systems. Validation of MBE was acomplished through a prototype tool used for generating a well known sinthetic *benchmark* and a *benchmark* specialized for a given application.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Entre os vários problemas relacionados ao gerenciamento de sistemas de informação, neste trabalho enfoca-se um dos mais centrais: a *análise de desempenho de sistemas de bancos de dados.* Das várias técnicas existentes, a metodologia aqui desenvolvida utiliza uma técnica conhecida popularmente por *benchmark de banco de dados.* A ênfase principal da metodologia está na caracterização do ambiente de execução, carga de trabalho e o banco de dados, para que sejam representativas de uma aplicação ou conjunto de aplicações do usuário.

1.1 Motivação

Atualmente, empresas e organizações dependem fortemente de *Sistemas de Informação* (SI) para gerenciar eficientemente seus negócios. Isso ocorre, principalmente, quando consideramos empresas de grande porte, porque além delas possuirem um patrimônio vultoso em termos de equipamentos e facilidades, que precisam ser convenientemente gerenciadas, elas também empregam um enorme número de funcionários que realizam várias atividades em conjunto. Entre outros fatores, a necessidade de *integração* entre os vários *níveis de decisão* existentes nas empresas levam naturalmente à construção e à utilização de SI. Tais níveis de decisão, aos quais nos referimos, abrangem desde os níveis considerados mais operacionais, relacionados às tarefas diretamente ligadas à produção, até os níveis estratégicos responsáveis pelo planejamento da empresa.

Nas empresas que prestam serviço público de telecomunicações, para citar um exemplo, é necessário, tipicamente, uma enorme quantidade de equipamentos (centrais telefônicas, cabos, tubulações, armários etc.) distribuídos geograficamente em uma determinada região para suportar os mais variados tipos de serviços de telecomunicações: telefonia, comunicação de dados, telex, etc.

Em uma empresa de telecomunicações como a Telesp[1] cerca de 25.000 funcionários gerenciam uma planta de mais de 2.700.000 terminais telefônicos [2]. Estes funcionários realizam inúmeras atividades que visam, basicamente, proporcionar serviços para os usuários, assim como administrar as operações da companhia.

Entre as várias atividades que estes funcionários realizam, podemos citar: o atendimento às reclamações dos usuários, envio de equipes de reparos, processamento de ordens de serviço, auxílio a usuários na escolha de serviços e no estabelecimento de chamadas telefônicas, elaboração de projetos e a contrução da infra-estrutura necessária para instalações de cabos e equipamentos, planejamento da disponibilidade centralizada de facilidades e teste de linhas privativas. Paralelamente devem ser mantidos os equipamentos instalados, as facilidades, os registros de assinantes (nome, endereço, número do terminal etc.) e os registros de utilização da rede (chamadas telefônicas realizadas, conexões a redes de dados etc.). Ainda, o provimento de informações estatísticas de tráfego e demanda (previsão) para planejamento de novas facilidades etc.

Todas essas atividades geram um volume muito grande de informações. A administração da empresa, por sua vez, deve dispor de **SI** adequados para manipular, armazenar e distribuir seletivamente estas informações entre as diferentes áreas que compõem a empresa: engenharia, operações, econômico-financeira e administrativa.

O bom funcionamento de um **SI**, por ser fundamental para a administração da empresa, deve exibir uma *qualidade de serviço* bastante elevada, de modo a garantir confiabilidade, integridade e disponibilidade das informações, assim como a redução do seu custo de geração e manutenção. Na prática essa qualidade está relacionada ao tempo de resposta das transações dos usuários, taxa de processamento de transações, confiabilidade contra falhas, capacidade de expansão e integração com outros sistemas.

Para que os **SI** sejam bem utilizados, os problemas acerca do gerenciamento dos dados, administração do sistema e técnicas de análise de desempenho, devem ser estudados. Para a análise de desempenho escolhemos uma técnica conhecida popularmente por *benchmark de banco de dados.*

1.1.1 Gerenciamento dos dados

Nos dias de hoje, observa-se uma crescente utilização de *Sistemas de Gerenciamento de Banco de Dados (SGBDs)* no suporte a **SI**. Esta popularidade deve-se não somente à variada gama de serviços que são oferecidos para o apoio ao desenvolvimento de sistemas, como também ao elevado estágio de desenvolvimento da tecnologia e ao alto grau de disseminação de metodologias de projeto de sistemas que facilitam a utilização dos SGBDs.

[1] Telecomunicações de São Paulo S/A.
[2] Dados de 1989 [RADE89]

Conceitualmente, um SGBD é um *software* genérico concebido para gerenciar grandes quantidades de dados. Este gerenciamento envolve tanto a definição das estruturas para o armazenamento das informações assim como a provisão de mecanismos para manipulá-las. Uma das grandes vantagens da utilização de um SGBD para o suporte a um **SI** está na capacidade de exercer um controle mais organizado dos dados e das aplicações (ou programas) que fazem acesso a esses dados. Para isso, o SGBD provê uma interface padronizada que facilita a manipulação dos dados pelos *programas*, permite o uso compartilhado do banco de dados por diversos usuários, que podem possuir diferentes visões do banco de dados, proporciona segurança e integridade das informações armazenadas no banco de dados e, mesmo no caso de queda do sistema ou de tentativas de acesso não autorizadas, deve preservar as informações. Além disso, se os dados forem compartilhados simultâneamente por diversos usuários, o sistema impede a possibilidade da ocorrência de resultados anômalos exercendo um controle de concorrência.

1.1.2 Gerenciamento do sistema

Os SGBDs possuem inúmeras facilidades para o desenvolvimento de sistemas. No entanto, o seu gerenciamento e manutenção ainda é considerado uma tarefa bastante árdua e complexa.

Nas grandes empresas, geralmente existem equipes formadas por especialistas de bancos de dados (gerentes de sistema, analistas de suporte) exclusivamente dedicadas à atividade de gerenciamento do SGBD. Estas pessoas, juntamente com os usuários responsáveis pelo desenvolvimento das *aplicações*, tomam, freqüentemente, um grande número de decisões que provocam, direta ou indiretamente, um impacto significativo sobre os fatores que influenciam a qualidade de serviço destes sistemas.

Uma primeira decisão a ser tomada com relação à utilização de um SGBD, se refere à escolha de quais aspectos das atividades da empresa devem ser representados no banco de dados. As decisões posteriores incluem desde a escolha do modelo de dados para representar as informações e a linguagem de desenvolvimento, até a escolha das estruturas físicas dos arquivos. A um nível mais abaixo, muitas outras decisões ainda são necessárias, como a escolha de índices para otimizar o custo de processamento das transações, a determinação dos tamanhos de blocos etc. Todas essas decisões guardam uma forte interação entre si e, se por um lado algumas delas podem parecer bastante intuitivas, em geral é muito difícil determinar qual a influência de tantas variáveis sobre o *desempenho do SGBD* [SEVC81].

Quando nos referimos ao desempenho de um SGBD existem basicamente dois aspectos que são de especial interesse: o *desempenho do usuário* [LOCH78] e o *desempenho do sistema* propriamente. O *desempenho do usuário* se refere a aspectos relativos à facilidade de utilização, nível da interface do usuário, aprendizagem da linguagem de programação etc. O *desempenho do sistema*, por

outro lado, se preocupa basicamente com a avaliação de fatores quantitativos tais como: o tempo de resposta, o *"throughput"* [3], ou número de referências lógicas e físicas à memória secundária, que procuram expressar a eficiência do sistema no atendimento de requisições dos usuários (transações). Neste trabalho, estamos interessados apenas neste último aspecto, ou seja, o desempenho do sistema. Os aspectos referentes ao desempenho do usuário, apesar de sua importância, não serão tratados aqui.

O *desempenho do sistema* é o resultado de uma complexa interação entre fatores *internos* e *externos* ao SGBD. Entre os *fatores internos* podemos citar:

- o grau de concorrência e compartilhamento do sistema.
- a carga de trabalho (conjunto de transações e perfil de freqüência de utilização).
- a disponibilidade de caminhos de acesso.
- a configuração adequada de parâmetros do sistema (tamanhos de blocos, de buffers etc.).
- o conteúdo, distribuição e disposição dos valores no banco de dados.
- as políticas de gerenciamento de memória.
- o *software* que implementa o SGBD propriamente, com seus módulos que são responsáveis pelas atividades de organização, manutenção do banco de dados, execução de transações etc.

Os *fatores externos* são aqueles que fazem parte do *ambiente computacional* no qual o SGBD está instalado e causam uma *carga de trabalho externa* ao sistema. Estas cargas de trabalho são compostas de programas que são executados concorrentemente ao SGBD e que disputam os mesmos recursos computacionais (e.g. memória, processador, discos etc.). Já que os SGBDs geralmente são utilizados em computadores não dedicados, estes fatores também devem ser analisados [TURB87].

Entre os fatores que influenciam o desempenho, em alguns deles, os usuário não possuem nenhum controle, tais como, a distribuição dos valores no banco de dados ou o código que implementa o SGBD propriamente. Por outro lado, algumas modificações como, a criação de índices e agrupamentos, e a definição de parâmetros físicos (tamanho de bloco, número de *buffers* etc.) do SGBD, devem ser realizadas pelo próprio usuário ou pelo gerente do sistema. Tais modificações devem ser realizadas de modo a otimizar a utilização dos recursos disponíveis no sistema.

No entanto, o maior problema com relação às otimizações que podem ser realizadas nos SGBDs, consiste em saber *como* e em *qual* extensão, essas modificações podem influenciar o desempenho do sistema globalmente. Para auxiliar

[3] Número de transações executadas por unidade de tempo (tps).

esta fase difícil, muitas propostas de metodologias têm sido apresentadas e muito progresso já tem sido alcançado, principalmente com relação ao projeto lógico de bancos de dados.

O desenvolvimento de projeto de bancos de dados orientado para o desempenho tem sido empregado em várias metodologias, sendo, até mesmo, algumas delas automatizadas. Tais metodologias produzem soluções consideradas boas ou até mesmo ótimas se forem obtidas sob determinadas premissas. No entanto, os critérios pelos quais se julgam estes projetos, muitas vezes são difíceis de serem relacionados à alguma *medida de desempenho* que seja considerada eficiente para gerentes de sistemas ou por usuários, tal como o tempo de resposta ou o *throughput* do sistema [SEVC81]. De modo geral, grande parte das metodologias já desenvolvidas enfocam com maior ênfase apenas a fase do *projeto conceitual* (ou projeto lógico de banco de dados), deixando de lado o *projeto físico*. Este último, no entanto, não é trivial e exige a utilização de ferramentas especializadas, porque é uma etapa onde existe uma quantidade muito grande de possíveis alternativas de solução.

Em suma, a *análise de desempenho* de sistemas de bancos de dados assume importância devido às seguintes razões:

- *Avaliação da relação custo* × *benefício*. A utilização de medidas de desempenho pode ser útil para a obtenção da relação custo × benefício em situações tais como: escolha das diversas alternativas de projeto, avaliação de um SGBD ou, ainda, de um *hardware* específico.

- *Avaliação da capacidade*. A análise de desempenho pode ser necessária para avaliar se uma determinada alternativa possui um nível de desempenho aceitável para uma determinada aplicação. Uma questão que sempre surge para um gerente de sistema é saber se a aplicação desejada pode ser executada convenientemente em um determinado SGBD instalado em uma determinada máquina.

Além disso, a preocupação com o desempenho torna-se crescente na medida em que existe uma grande proliferação de SGBDs no mercado. A cada dia surgem novas ofertas assim como versões mais atualizadas dos sistemas já existentes. Somente para se ter uma idéia deste número, se citarmos apenas aqueles produtos que são considerados como os mais populares encontramos: ORACLE, INGRES, DB2, INFORMIX, UNIFY, SUPRA, IDMS, ADABAS, IMS, SQL/DS, DMSII, RDB, entre outros. Mesmo se considerarmos uma mesma máquina e sistema operacional, podem ser encontradas várias opções disponíveis.

Esta grande disponibilidade de SGBDs, por um lado, favorece o usuário na medida em que ele adquire um poder de barganha maior em relação aos vários fornecedores; por outro lado, também aumenta a expectativa quanto ao desempenho destes sistemas e torna crescente a necessidade de estabelecimento de *padrões de avaliação*. De um ponto de vista ideal, por exemplo, um gerente de

sistema gostaria de poder selecionar um SGBD para uma aplicação, digamos, tipicamente transacional, apenas comparando as opções existentes e escolhendo aquela que possuísse a melhor relação custo x benefício, obtida a partir das características técnicas de *throughput* (tps) ou tempo de resposta fornecidas pelo próprio fabricante, sem a necessidade de ter que fazer testes nos possíveis candidatos [ANON85]. Infelizmente, este enfoque ainda é inviável na prática, pois, na maioria dos casos, os fornecedores utilizam *metodologias* de análise próprias, não padronizadas, onde procuram valorizar apenas as características mais favoráveis de seus produtos. Conseqüentemente, a comparação entre sistemas de diferentes fornecedores ou, mesmo, entre sistemas com diferentes capacidades, torna-se bastante difícil.

Apesar do desempenho do sistema ser um item que sempre consta na avaliação de qualquer sistema, não se deve, no entanto, tomá-lo como o fator predominante no processo de escolha de um SGBD. Deve ser claro, porém, que as decisões quanto à escolha de um sistema devem estar baseadas em algum critério técnico e não apenas no conhecimento e "gosto" dos decisores por um determinado sistema, visto que o custo destes sistemas não é nada desprezível [4]. Sendo assim, outros fatores, tais como: nível da interface, facilidades de utilização, compatibilidades com outros sistemas, flexibilidade etc. também devem ser observados [STON85].

1.1.3 Técnica de *benchmark*

As várias técnicas utilizadas para análise de desempenho, geralmente, recaem em um dos três tipos de modelos clássicos: *modelos analíticos*, *modelos de simulação* e *modelos experimentais*. Neste trabalho utilizamos uma técnica experimental chamada de *técnica de benchmark*. Esta técnica tem sido uma das mais utilizadas para avaliação de desempenho de SGBDs, a tal ponto, de já existirem alguns *benchmarks* que podem ser considerados praticamente como verdadeiros padrões [ANON85], [BITT83].

Genericamente, a *técnica de benchmark de sistemas* consiste na execução de um conjunto conhecido de programas sobre um sistema computacional para se avaliar o seu desempenho. Pelo fato de ser uma técnica experimental ela depende da maturidade da tecnologia envolvida. Ao contrário, as técnicas que utilizam modelos de simulação e modelos teóricos podem ser empregadas em sistemas ainda nascentes, onde ainda não se têm dados operacionais disponíveis. Dentro do contexto de qualquer nova tecnologia, a técnica de *benchmark* deve ser a última a poder ser utilizada, já que ela necessita que todos os componentes do sistema já tenham sido implementados e testados.

No *benchmark de bancos de dados* um conjunto de transações, ou a carga de trabalho como é comumente denominada, é executada sobre um banco de dados conhecido. Tanto a carga de trabalho quanto o banco de dados podem ser *reais*

[4] Calcula-se que o custo de SGBDs seja acima de US$ 300.000 para sistemas de médio e grande porte.

ou *sintéticas* (artificiais). Além disso, as medidas geralmente utilizadas para análise de desempenho são medidas macroscópicas tais como tempo de resposta ou *tps*.

Segundo Turbyfil [TURB87], com o desenvolvimento da tecnologia de bancos de dados relacionais nestes últimos anos, atingiu-se um ponto que justifica e, ao mesmo tempo, necessita de testes extensivos de desempenho utilizando-se *benchmark*.

Em termos globais, a realização de um *benchmark* para auxiliar na escolha de SGBDs exige um custo relativamente elevado. Isso se deve não somente ao tempo de máquina utilizado, como também ao custo do pessoal alocado ao processo. Desta forma, para que a aplicação da técnica de *benchmark* possa ser bem sucedida, deve haver um planejamento antecipado, detalhando cada uma das etapas: concepção, coleta de dados, construção do banco de dados de teste, conjunto de transações e finalmente o roteiro de execução dos testes, de modo que a execução do *benchmark* seja realizada dentro do tempo planejado, minimizando eventuais interrupções dos sistemas em produção e evitando, assim, desperdícios de recursos.

1.2 Proposta de trabalho

Neste trabalho desenvolvemos uma *Metodologia de Análise de Desempenho baseada em Benchmark Especializado*. Esta metodologia engloba as etapas de concepção, coleta, organização e construção do *benchmark* (banco de dados e carga de trabalho). O usuário parametriza o *benchmark* obtido de acordo com as características de uma aplicação ou de um conjunto típico de aplicações. O banco de dados e a carga de trabalho são convenientemente modeladas e posteriormente geradas automaticamente por programas que implementam a metodologia. Os vários objetos que modelam a aplicação são armazenados em um *dicionário de dados* centralizado, que representa o *Modelo do Sistema de Bancos de Dados*. Este modelo é sub-dividido em um *Modelo de Representação dos Dados* e um *Modelo de Carga de Trabalho*.

No *Modelo de Representação dos Dados*, o usuário define as tabelas (relações), fornece informações sobre tipo de dados utilizados, número de atributos, distribuição de valores, número de valores distintos etc. complementando as informações obtidas inicialmente na fase de modelagem do banco de dados. A fase de modelagem de dados, por sua vez, utiliza um *diagrama de entidades e relacionamentos* [CHEN77], onde é informado o esquema do banco de dados.

O *modelo da carga de trabalho* é representado por um *modelo de execução* juntamente com um *modelo de descrição* de transações. No *modelo de execução* são representados hierarquicamente os vários níveis de concorrência presentes no sistema que se deseja modelar: *ambiente externo* ao SGBD, *ambiente interno*, *aplicação* e *transação*. O *nível externo* especifica a carga externa ao SGBD representada por processos que são executados concorrentemente no mesmo sistema

computacional. O *nível interno* e o *nível aplicação* especificam respectivamente o número de usuários utilizando diferentes aplicações simultâneamente no SGBD e o número de usuários utilizando uma mesma aplicação. No *nível de transação* podem ser especificadas as freqüências de execução de cada tipo específico de transação, obtendo-se, assim, um perfil de utilização. O *Modelo de Descrição de Transações* baseia-se em um *paradigma* de utilização do banco de dados através de formulários ou janelas. Neste paradigma as transações são centradas em uma *visão base* associada à janela sobre a qual realiza-se as interações e, paralelamente, outras operações são realizadas sobre outras tabelas relacionadas à visão base. A adoção deste paradigma é justificada pela sua representatividade e pela naturalidade da sua utilização pelo usuário que torna mais facil a obtenção do perfil das transações que são solicitadas na aplicação.

A descrição do sistema é composta por vários objetos de modelagem (relações, atributos, domínio de valores, aplicações, transações etc) que são organizados e armazenados em um banco de dados em memória secundária (*dicionário de dados* do sistema). Na medida em que a ferramenta necessita de um objeto de modelagem, durante sua execução, os dados correspondentes a este objeto são recuperados do dicionário. A partir destas informações é possível obter automáticamente, através de programas, o esquema do banco de dados, o seu conteudo (valores para as tuplas das relações do banco de dados) e a carga de trabalho que mais se aproximem da aplicação, ou tipo de aplicações, do usuário.

A metodologia, no entanto, não prevê um suporte específico para análise de alternativas de projeto, tal como a escolha de índices. Em contrapartida, ela oferece a possibilidade de construção de um ambiente facilmente gerado pelo usuário, onde se podem testar as diferentes alternativas de projeto de bancos de dados a serem avaliadas pela técnica de *benchmark*.

Para a validação da metodologia foi construída uma ferramenta protótipo, que pode ser utilizada tanto para selecionar um SGBD através da análise de desempenho, como para auxiliar um projetista de aplicação ou um gerente de sistema a escolher uma boa alternativa de projeto para uma dada configuração de dados e transações. As altenativas podem ser avaliadas em um ambiente de testes, composto de um banco de dados e carga de trabalho, geradas rapidamente pela ferramenta e executadas sobre o SGBD.

A utilização desta ferramenta proporciona facilidades de gerar dados com um grau de complexidade bastante elevado, próximo à situações reais. Relações com chaves primárias simples, compostas e relacionamentos de qualquer cardinalidade (1:1, 1:N, M:N) podem ser obtidos entre as relações construídas. Desta forma, os usuários têm a possibilidade de formular transações razoavelmente complexas que podem incluir *joins* entre relações, visões etc.

O *benchmark* obtido pela ferramenta é todo expresso na linguagem SQL, o que permite o seu transporte para diferentes SGBDs. Mesmo considerando a existência de soluções mais eficientes para a construção do *benchmark*, esta solução foi adotada por estar de acordo com os padrões de mercado e facilitar a sua portabilidade entre diferentes sistemas.

Não fez parte deste trabalho a análise dos resultados produzidos pelos *benchmarks* gerados. Concentramos, principalmente, no desenvolvimento da metodologia (MBE) e construção de uma ferramenta que implementa a metodologia.

1.3 Organização da tese

Este trabalho divide-se em 7 capítulos. O Capítulo 1 introduziu o trabalho, mostrando o contexto, principais motivações e objetivos. O Capítulo 2 faz uma apresentação dos conceitos básicos que serão utilizados ao longo do trabalho. O Capítulo 3 traz um resumo dos trabalhos mais recentes publicados na área de *benchmark* de bancos de dados, com especial enfoque sobre algumas propostas que vêm se tornando verdadeiros padrões de avaliação de SGBDs. Ao final deste capítulo, apresentamos um lista de artigos contendo resultados de análise de desempenho de vários SGBDs, que foram colecionados ao longo da elaboração deste trabalho. O Capítulo 4 apresenta a *Metodologia de Benchmark Especializado* que foi desenvolvida. No Capítulo 5 discute-se a implementação da ferramenta protótipo e algumas justificativas de escolhas realizadas. No Capítulo 6, é feita uma avaliação dos resultados obtidos pela ferramenta. Finalmente, o Capítulo 7 apresenta as conclusões obtidas do trabalho seguida da bibliografia utilizada. Os apêndices contêm as especificações de alguns testes e resultados.

Capítulo 2

Conceitos básicos

Sistemas de gerenciamento de bancos de dados (SGBD) são concebidos para gerenciar grandes quantidades de informação. O gerenciamento dos dados envolve tanto a definição de estruturas para o armazenamento da informação como o fornecimento de mecanismos para manipulá-la. O sistema deve proporcionar segurança das informações armazenadas no banco de dados contra quedas eventuais do sistema ou tentativas de utilização não autorizadas. Ainda, deve permitir o compartilhamento dos dados, evitando possíveis erros de integridade devido à utilização simultânea.

A importância da informação na maioria das empresas tornou o SGBD um recurso valioso, e isso tem levado ao desenvolvimento de vários conceitos e técnicas, destinados ao eficiente gerenciamento do sistema, que apresentamos a seguir.

2.1 Sistemas de bancos de dados

Na parte central de um sistema de informações, geralmente, reside um *sistema de bancos de dados*. Este sistema pode ser visualizado como a combinação de:

- *Software* específico de gerenciamento dos dados (SGBD).
- Um conjunto de programas de aplicação.
- *Banco de dados*.
- *Sistema computacional - hardware* do computador e o sistema operacional.

Um *banco de dados*, ou base de dados, é uma coleção organizada de dados operacionais pertencentes a um sistema de informação e que podem ser utilizados por vários indivíduos dentro de uma ou mais organizações. Em sistemas

corporativos o banco de dados armazena um grande volume de informações sendo medido, tipicamente, em termos de *gigabytes*[1] de dados.

Um SGBD, por sua vez, é uma ferramenta de *software* genérica destinada a definição, manutenção, análise e manipulação de um banco de dados. Um de seus objetivos principais é proporcionar aos usuários uma *visão abstrata* dos dados. Normalmente, considerações quanto à *eficiência* no armazenamento da informação levam à concepção de estruturas de dados complexas no banco de dados. No entanto, como os usuários não são necessariamente especialistas em computação, eles poderiam ter dificuldades em lidar com estas estruturas. O SGBD, por sua vez, deve "esconder" esta complexidade, permitindo que os usuários possam visualizar os dados em um nível mais abstrato sem a preocupação de serem forçados a compreender os detalhes das estruturas físicas que armazenam os dados e nem de como são manipuladas ou mantidas.

Durante sua existência, os sistemas de bancos de dados vão se modificando, na medida em que as informações são inseridas, modificadas ou removidas. Em um dado instante de tempo, chamamos de uma *instância* do banco de dados a coleção de informações que estão armazenadas e que se modifica ao longo do tempo. Porém, a concepção global do sistema (*esquema* do banco de dados) não se modifica assim tão freqüentemente. Basicamente, os esquemas são estruturas de representação dos dados que contém as definições dos objetos que serão representados no banco de dados e suas respectivas ligações (relacionamentos). Se fizermos uma analogia com as linguagens de programação, os esquemas são como os tipos de dados e as variáveis, juntamente com os valores que assumem, são suas instâncias.

Uma das principais funções de um SGBD, no sentido de oferecer uma visão abstrata dos dados para o usuário, é a *independência dos dados.* Essa independência pode ser classificada como sendo de dois tipos:

- *Independência física*: capacidade de modificar o esquema físico sem precisar reescrever os programas de aplicação. Ocasionalmente, mudanças no nível físico são necessárias para melhorar o desempenho do sistema.
- *Independência lógica*: capacidade de modificar o esquema conceitual sem a necessidade de reescrever os programas de aplicação.

A independência de dados lógica é mais difícil de se conseguir do que a independência física, uma vez que os programas ainda são muito dependentes da estrutura lógica dos dados que eles acessam. Em especial, os SGBDs baseados no paradigma de *objetos complexos* (ou SGBDs orientados a objetos) utilizam a característica de *encapsulamento* dos dados para promover maior independência lógica. Através deste paradigma, somente os procedimentos internos à definição de um objeto podem manipular suas estruturas de dados próprias, assim, conseguindo-se esconder detalhes de implementação dos usuários. A utilização dos objetos pelos usuários somente é consentida através de uma interface

[1] Bilhões de bytes

que se comunica com o código que implementa os procedimentos internos, autorizados a manipular os objetos, garantindo-se assim, sua integridade [ZDON90].

Um esquema de banco de dados é especificado por um conjunto de definições que são expressas em uma linguagem especial, chamada *linguagem de definição dos dados* (DDL)[2]. Como resultado da compilação de instruções da DDL tem-se um conjunto de tabelas que são armazenadas em um arquivo especial chamado *dicionário de dados.*

Para a manipulação dos dados no banco de dados, o SGBD provê uma *linguagem de manipulação dos dados* (DML)[3] que permite a recuperação, inserção, atualização e a remoção de informações armazenadas no banco de dados. Existem dois tipos de DMLs, as *procedimentais* e *não procedimentais*:

- *Procedimentais*: o usuário deve especificar quais dados são desejados e "como se deve chegar" até eles.
- *Não procedimentais*: o usuário especifica quais dados deseja "sem necessitar especificar como chegar" até eles.

As linguagens procedimentais exigem um conhecimento maior de conceitos de computação (programação) por parte do usuário e, por essa razão, são mais difíceis de serem aprendidas. Elas são mais indicadas para os usuários especialistas que desenvolvem as aplicações. Em contrapartida, as linguagens não procedimentais possuem um nível de complexidade mais baixo e, portanto, são mais facilmente aprendidas por usuários não especialistas.

A linguagem SQL[4] é uma linguagem não procedimental de acesso a bancos de dados, padronizada pela ISO e ANSI (comitê X3H2). Trata-se de uma linguagem *english-like* que, apesar de possuir no nome a expressão *query language*, engloba todas as funções de uma DDL e uma DML. Inicialmente a SQL foi desenvolvida para o Sistema R [ASTR76] e posteriormente passou a ser utilizada em vários SGBDs, tais como: SQL/DS [KORT89], DB2 [DATE86b] e ORACLE [DIMM86], entre outros.

Os usuários e os programas de aplicação interagem com o SGBD através de *transações* [5]. Basicamente, uma transação é um seqüência lógica de comandos que devem ser tratados como uma unidade lógica de trabalho, ou seja, o SGBD deve garantir que todas as operações de uma transação devam ser terminadas por completo (i.e. todas as alterações devem ser realizadas permanentemente no banco de dados) ou então que nenhuma das operações deva ser efetivada, sob pena de perda de *integridade dos dados.* Se o sistema ou programa de aplicação do usuário falhar enquanto estiver executando uma transação, então o banco de dados deve ser restaurado para o estado anterior ao início da transação.

[2] *Data Definition Language*

[3] *Data Manipulation Language*

[4] Structured Query Language.

[5] Ao longo deste trabalho serão os mesmos os conceitos de transação, requisição ou operação no SGBD.

Conceitualmente, todo SGBD possui um *modelo de dados* pelo qual se segue toda a sua filosofia de utilização. Entre os modelos de dados mais conhecidos destacam-se: modelo em redes (CODASYL), hierárquico, relacional e, mais recentemente, o modelo orientado a objetos. A seguir mostramos alguns exemplos de SGBDs que implementam estes modelos:

- **rede**: DMS [Sperry Univac 73], IDMS [Cullinane 75], IDSII [Honeywell 75]
- **hierárquico**: IMS [IBM 78], System 2000 [MAGA81].
- **relacional**: Sistema R [ASTR76], INGRES [STON76], ORACLE [DIMM86].
- **orientado a objetos**: DAMOKLES [ABRA88], POSTGRES [STON86].

Normalmente, cada modelo possui uma nomenclatura própria para descrever os vários objetos que compõem um banco de dados. Portanto, para facilitar o entendimento ao longo deste trabalho, adotamos o modelo relacional como meio de representação do banco de dados. Neste modelo, o *banco de dados* é representado por meio de um conjunto de tabelas (ou relações), onde cada tabela é composta de linhas (ou tuplas) e de colunas (ou atributos). O conceito de uma tabela envolve noções bastante simples e intuitivas, que facilitam o usuário a definir seu banco de dados. Além disso, existe uma correspondência direta entre o conceito de tabela e o conceito matemático de *relação* [KORT89]. Não obstante, o modelo de dados relacional não é um pré-requisito da metodologia aqui apresentada, que acreditamos possa ser empregada para qualquer outro tipo de modelo de dados.

Além da possibilidade de implementação de tabelas, o modelo relacional possui um mecanismo bastante poderoso para a representação de *visões* dos usuários. Uma visão, do ponto de vista conceitual, é uma relação virtual - i.e. uma relação que não "existe" realmente, mas, do ponto de vista do usuário, comporta-se como se existisse - sobre a qual pode ser definida qualquer operação relacional. Na realidade, as visões não são diretamente suportadas pelo SGBD, elas são apenas "definidas" em termos de outras relações ou ainda de outras visões já definidas anteriormente. São estas definições que ficam armazenadas no catálogo de dados do sistema e que são utilizadas quando alguma operação é realizada sobre uma visão. No momento da sua utilização, a definição é processada e a visão se "materializa" em tempo de execução pelo próprio SGBD. De acordo com sua definição, qualquer operação válida sobre uma relação também poderia ser realizada, teoricamente, sobre uma visão. Na prática, entretanto, os SGBDs não conseguem implementar esta característica genericamente e, em muitos casos, existem restrições quanto às operações que realizam atualizações e remoções de tuplas em visões.

2.2 Projeto de bancos de dados

O *projeto de bancos de dados* pode ser considerado como a disciplina de desenvolver sistemas de bancos de dados a partir dos requisitos dos usuários. Um conceito central para o projeto de bancos de dados consiste na *modelagem dos dados*. Basicamente, a modelagem refere-se à busca de uma representação para os requisitos dos usuários com relação a forma e estrutura dos dados para o melhor armazenamento das informações desejadas e a descrição dos requisítos de processamento destas informações. As propriedades estáticas dos objetos e os relacionamentos entre eles são representadas em termos de estruturas de dados, enquanto que as propriedades dinâmicas, das ações que podem ser realizadas sobre estes objetos, são representadas em termos de processos.

Na literatura de bancos de dados se consideram, tradicionalmente, pelo menos duas etapas para a elaboração de um projeto de bancos de dados: *projeto lógico* de banco de dados e o *projeto físico*. O *projeto lógico* se preocupa com a representação dos requisitos do usuário através de modelos abstratos que procuram facilitar a formalização de uma aplicação. Nesta etapa o usuário não deve se preocupar com detalhes de como estes modelos abstratos serão representados fisicamente no SGBD. O *projeto físico*, por sua vez, se preocupa com a seleção das melhores estruturas e operações para a representação do modelo lógico obtido no passo anterior. Decisões sobre escolhas de caminhos de acessos, agrupamento[6] etc. também fazem parte desta etapa. Complementando esta fase, as escolhas definidas devem estar de acordo com as restrições que, geralmente, os SGBDs impõem.

O processo de projeto de bancos de dados é uma atividade extremamente árdua e complexa dado a grande variedade de soluções possíveis para um determinado problema. Neste sentido, a utilização de uma metodologia pode auxiliar significativamente o projeto de bancos de dados.

Na metodologia proposta por Chen [CHEN77], *Modelo de Entidades e Relacionamento* (MER), a percepção do universo é baseada em termos de objetos e relacionamentos entre estes objetos. Este modelo permite visualizar o projeto de banco de dados através de um esquema que representa a estrutura lógica dos dados. Pode-se dizer, também, que este modelo captura as características estáticas da aplicação.

Na representação de propriedades dinâmicas a metodologia de Gane [GANE79] é bastante utilizada. Esta metodologia está centrada na descrição dos processos que manipulam as informações do banco de dados. Fazem parte desta metodologia elementos como processos, entidades externas, depósitos de dados e fluxos de dados. Os fluxos de dados podem ocorrer entre processos, entre processos e entidades externas ou depósitos de dados. Pode-se dizer que se trata de uma metodologia centrada ou *orientada para processos*, ao contrário do MER, que é *orientado para dados*. O esquema de DFD, proposto por Gane, procura facilitar

[6] proveniente do termo *clustering*.

a captura de detalhes de representação das transações dos usuários utilizando uma descrição dos processos organizadas em níveis de abstração, sugerindo uma melhor estrutura para o sistema.

2.3 Representação de sistemas de bancos de dados

Com vistas a facilitar a análise de desempenho, é conveniente adotar um modelo de representação do sistema de bancos de dados estruturado em *níveis de abstração* [SEVC81]. Estes níveis vão desde os mais abstratos até aqueles que procuram representar mais proximamente as características físicas do sistema, tais como as estruturas de dados e o hardware no qual o sistema está instalado. Uma das vantagens em se utilizar uma representação estruturada em níveis advém da facilidade de se permitir a escolha de determinadas alternativas sem que sejam afetadas as escolhas já realizadas em níveis superiores.

Na literatura de análise de desempenho existem várias propostas de definição de níveis de abstração para a representação de sistemas de bancos de dados. Aqui apresentamos a proposta de Raposo [RAPO86], que é derivada basicamente de Sevcik [SEVC81].

A ferramenta de análise de sistemas de bancos de dados PROPHET apresentada em [RAPO86] baseia-se em uma proposta estruturada em níveis de abstração, onde, se utiliza modelos teóricos (teoria de filas) no nível mais baixo. Este modelo de representação divide-se em 4 níveis: *semântico, esquema, interno, sistema hardware.* Estes níveis vão desde o mais abstrato (semântico) até o mais concreto (sistema hardware). Para cada decisão de projeto tomada em um nível realizam-se transformações para uma informação mais concreta no nível abaixo mais próximo, procurando-se fornecer uma representação para projeto de banco de dados baseando-se em uma abordagem orientada para desempenho do sistema como um todo.

Em cada nível desta estrutura são realizadas as seguintes funções:

- *Nível Semântico*: onde são decididos quais aspectos da atividade da organização devem ser representados no banco de dados. Este processo de escolha em muito se beneficia do uso de metodologias de desenvolvimento. Ao final, obtém-se um modelo lógico do sistema.

- *Nível de Esquema*: trata da representação do modelo lógico, obtido no nível anterior, através da escolha de estruturas adequadas, relações e seus atributos, que melhor representem este modelo. A representação obtida no nível de esquema deve ser compatível com o modelo de dados disponível no SGBD, levando-se ainda em consideração possíveis restrições impostas pelo SGBD.

- *Nível Interno*: onde devem ser escolhidas estruturas de dados para dar suporte às atividades desejadas. Neste nível podem ser realizadas escolhas de índices, estratégias de acesso aos dados, particionamento dos dados nas unidades de armazenamento, escolha de políticas de gerenciamento de buffers, codificação de dados etc. As soluções escolhidas nesse nível devem ser o máximo possível independentes de considerações do sistema hardware.

- *Nível Sistema/Hardware*: fica na base desta hierarquia e envolve a maior parte das definições ligadas à máquina propriamente. Estas decisões envolvem: obtenção de disciplinas de *schedulling*, gerenciamento de memória, concorrência com outros processos externos ao SGBD, especificação de subsistema de comunicação etc. Para a representação física deste nível utiliza-se modelos baseados em teoria de filas.

Em uma modelo de representação do sistema de bancos de dados estruturado desta forma, existe a facilidade de compreender a interação das diversas decisões que são tomadas durante o projeto de um banco de dados. Os vários fatores que influenciam o desempenho do sistema possuem interações bastante complexas. Em cada um dos níveis de abstração pode existir uma variedade muito grande de escolhas e cada uma pode afetar significativamente o desempenho. O usuário pode definir cada característica de seu sistema e testar as várias escolhas possíveis em relação ao desempenho observado do sistema.

2.4 Análise de desempenho

A análise de desempenho em sistemas de banco de dados é uma tarefa necessária no gerenciamento de um SGBD. Muitas vezes é importante avaliar-se a relação custo x benefício proporcionada por alguma escolha, que pode envolver tanto a definição de um *hardware* quanto a escolha de um *software*. Em outras ocasiões, é importante ser avaliada a *capacidade* de um sistema. Por exemplo, uma situação bastante comum ocorre quando se deseja saber se um determinado SGBD instalado em uma determinada máquina é capaz de suportar eficientemente a aplicação desejada, levando-se em consideração todo o ambiente de *hardware* e *software* existente.

Tradicionalmente, na análise de desempenho se utilizam os seguintes modelos:

- *Modelo Analítico*: baseia-se na obtenção de um conjunto de equações juntamente com os algoritmos para resolve-las, que relacionam medidas de desempenho à parâmetros do sistema. Usualmente empregam-se várias hipóteses teóricas a respeito do conteúdo do banco de dados, colocação dos registros nos arquivos, e comportamento da carga de trabalho de modo a

simplificar o modelo obtido. O custo econômico de sua aplicação é relativamente baixo, dado a facilidade de ser utilizado e a rapidez com que são obtidos os resultados.

- *Modelo de Simulação*: procura *reproduzir* as atividades do sistema de banco de dados de acordo com um conjunto de hipóteses e condições, eliminando a necessidade de experimentação no próprio sistema. Para produzir resultados precisos o modelo de simulação deve crescer em complexidade e isso requer um conhecimento maior e mais completo dos valores dos parâmetros que descrevem o sistema. Em alguns casos, no entanto, não é possível ou então é muito difícil obter estes valores; por exemplo, durante os estágios primários de desenvolvimento de um sistema, em que os dados não são disponíveis, ou mesmo durante sua operação, em que muitas vezes são difíceis de serem obtidos sem afetar o sistema em produção.

- *Modelo Experimental*: procura utilizar o próprio sistema de banco de dados para se obter os resultados. Alguns experimentos podem ser realizados utilizando-se cargas de trabalho sintéticas sobre um banco de dados também sintético [BITT83] através de uma técnica popularmente conhecida por *benchmark*. Ainda, os modelos experimentais podem envolver a monitoração de um sistema em produção para se obter valores reais [MAGA81].

Uma das maiores vantagens do modelo experimental, consiste na fidelidade alcançada nos resultados, pois se utiliza o próprio sistema (SGBD) durante os testes. No caso de monitoração, por exemplo, o trabalho envolvido é geralmente considerado muito grande e o desenvolvimento da ferramenta de apoio à análise de desempenho torna-se, na maioria das vezes, específico para aquele sistema em análise; ao contrário da técnica de *benchmark* que, em geral, permite que a mesma ferramenta possa ser empregada em diferentes SGBDs.

Apesar de existirem ferramentas próprias de avaliação estatística do sistema, disponíveis na maioria dos SGBDs, nem sempre é possível utilizá-las para a comparação de diferentes sistemas devido à ausência de padronização entre elas [DEWI85].

Em alguns casos é possível utilizar *modelos híbridos*, envolvendo mais de um modelo (em geral apenas dois). Estes modelos procuram associar as vantagens de cada modelo distinto de forma a obter uma representação do sistema mais realista ou, ainda, procurando facilitar a execução das ferramentas assim criadas. Em [EFFE84] utilizam-se modelos experimentais para a obtenção de dados através de um monitoramento do sistema real e modelos de simulação para a avaliação de possíveis alternativas de gerência de *buffers-pool*.

A tabela 2.1 faz uma comparação entre os vários modelos.

Na tabela, o sinal de + indica que a característica é favorável ao método, o sinal de − indica uma característica desfavorável e o sinal de +− indica que existem bons e maus aspectos do método.

2.5 Assimetria na utilização de recursos

Durante a observação da utilização de recursos computacionais, módulos de programas, posições de memória etc. freqüentemente se constata um comportamento assimétrico. Tal fato, há muito tempo vem sendo verificado em Ciência da Computação assim como em outras áreas [WILM89], [MAGA81]. Os estudos sobre assimetria são importantes para a área de desempenho de bancos de dados, quando se estuda o problema da otimização de transações [FINK88], [SALZ89], [MANN88], [WILM89], [SELI79], [MOHA89], ou a construção de modelos de análise de desempenho ou carga de trabalho [CHRI84], [MAGA81], [YU89].

Geralmente, a utilização assimétrica é referenciada como a *Regra de 80/20* [KNUT73], que significa que apenas cerca de 20% dos recursos são utilizados em 80% do tempo de utilização. Podem existir, inclusive, casos de assimetria mais acentuada, onde a regra se transforma em 90/10.

2.5.1 A distribuição de Zipf

G. K. Zipf foi o primeiro a observar e relatar o comportamento de distribuições assimétricas na utilização de recursos [WILM89]. Observando este tipo de comportamento, por exemplo, com a utilização de palavras[7], em populações de cidades, nomes de pessoas, etc, ele pôde formular uma lei que descrevia este comportamento.

A *Lei de Zipf*, como passou a ser chamada, pode ser enunciada da seguinte forma:

> "O *ranking*[8] das palavras em um texto é inversamente proporcional à sua freqüência de ocorrência".

O elemento mais freqüente recebe a posição do topo do *ranking*, ou seja, o valor numérico 1, e o menos freqüente recebe a última posição. Podemos, ainda, expressar a lei de Zipf matematicamente como:

$$rank_i \times \omega_i = \frac{1}{constante} \tag{2.1}$$

O valor ω_i é a freqüência do i-ésimo elemento e a $\sum_{i=1}^{n} \omega_i = 1$, onde n é igual ao número de valores distintos. O valor da *constante* pode ser obtido pelo n-ésimo número harmônico:

$$H_n = \sum_{k=1}^{n} \frac{1}{k} \tag{2.2}$$

[7] Ex. no livro *Ulysses* de James Joyce existem 260.430 palavras, mas, apenas as 135 palavras mais freqüentes, em um total de 26899 palavras distintas, correspondem a 50% da contagem total de palavras.

[8] O *ranking* é uma ordenação de acordo com a freqüência de ocorrência de cada elemento distinto.

Que pode ser aproximado por:

$$H_n = \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4}$$

Onde: $\gamma = 0.5772156649$ é a constante de Euler com 10 dígitos significativos [KNUT73].

Uma distribuição de Zipf mais geral pode ter bastante utilidade. Sua expressão é dada por:

$$rank^z \times \omega = \frac{1}{constante}$$

Em uma distribuição do tipo Zipf, z é chamado o fator de decaimento. Dependedo deste valor a distribuição pode se tornar mais ou menos assimétrica. Podemos verificar que quando $z = 0$ a distribuição é uniforme e quando $z = 1$ a distribuição é Zipf. Em uma distribuição do tipo Zipf o valor da constante pode ser calculada pelo e-nésimo harmônico de ordem z:

$$H_n^{(z)} = \sum_{k=1}^{n} \frac{1}{k^z}$$

Na Tabela 2.2 é mostrado os valores de freqüência para 10 valores distintos em distribuições do tipo Zipf com fatores de decaimento z iguais a 0, 0.5, 1 e 3 (valores extraídos de [TURB87]). Pode-se observar que a distribuição se torna mais assimétrica na medida em que z aumenta.

A utilização de distribuições assimétricas em modelagens de bancos de dados é muito importante pois, se assumimos apenas comportamento uniforme, isso pode nos levar a uma estimativa incorreta da seletividade de um predicado [CHRI84]. Em geral, quando a seletividade real de uma amostra é bastante pequena, mas os valores são distribuídos não uniformemente, a seletividade estimada, supondo um comportamento uniforme, tende a ser sobre-estimada o que, na prática, pode resultar na escolha de uma busca seqüencial na relação, quando neste caso o uso de um índice seria mais apropriado. De forma inversa, quando a seletividade real de um segmento de valores é elevada, mas os valores possuem uma distribuição muito assimétrica, a seletividade estimada, da mesma maneira que a anterior, tende a ser sub-estimada, o que pode resultar na escolha de um índice ao invés de uma busca seqüencial que seria mais conveniente neste caso. Portanto, pode-se concluir que a determinação incorreta da seletividade prejudica em muito a execução de operações no SGBD e, particularmente, a otimização em operações de *join* [TURB87]. Na seção seguinte veremos como a melhor estimativa da seletividade pode contribuir para a otimização de transações em um SGBD.

2.6 Otimização de transações

Em sistemas que dispõem de linguagens de consulta e de manipulação de dados em alto nível, tal como ocorre com o SQL, por exemplo, as requisições (ou transações) dos usuários são realizadas de maneira *não-procedimental*, sem que sejam feitas referências a *caminhos de acessos*. Isso obriga que estes sistemas implementem módulos chamados *otimizadores de transações*, que determinam o melhor caminho de acesso para a resolução das requisições do usuário.

A seguir fazemos um breve resumo sobre os conceitos que envolvem a otimização de transações e sua importância no desempenho dos SGBDs.

2.6.1 Função do otimizador de transações

Durante a última década, os SGBDs baseados no modelo relacional passaram da experiência dos laboratórios e ingressaram no campo comercial. Uma das grandes forças que impulsionaram estes sistemas é sua facilidade de uso. Usuários interagem com o sistema em um modo mais natural, utilizando linguagens de acesso não procedimentais, onde somente se especifica "o quê" deve ser recuperado e não "como" obter. Os comandos especificam que tabelas devem ser accessadas e quais as condições desejadas. Eles não especificam caminhos de acesso (i.e. índices) utilizados para obter os dados de cada tabela ou ainda a seqüência de quais tabelas devem ser acessadas e qual algoritmo utilizar. Desta forma, comandos relacionais podem ser executados independentemente da existência ou não de índices.

Pesquisadores que desenvolvem SGBD relacionais argumentam que os sistemas podem ser capazes de realizar decisões muito boas para as requisições dos usuários baseados em modelos de representação dos bancos de dados e fórmulas que estimam o custo de execução de diferentes alternativas de acesso [CODD82], [DATE86c]. Módulos de software chamados *otimizadores* fazem estas decisões baseados em modelos estatísticos do banco de dados[9]. Eles analisam e constrõem alternativas, escolhendo aquela que possui o menor custo de execução [FINK88], ou, ainda, a alternativa que apresenta uma solução que esteja dentro de limites pré-determinados de tempo de resposta, uso de recursos computacionais e custo de otimização (solução semi-otimizada) [SILV90].

2.6.2 Processamento de um comando SQL

Um comando em SQL para ser executado não requer que o usuário especifique nada sobre caminhos de acesso para que essa informação possa ser utilizada durante o seu processamento. Tipicamente, um comando em SQL é submetido a quatro fases de processamento: *parsing*[10], *otimização*, *geração de código* e

[9] As idéias contidas nesta seção, apesar de retratarem SGBD relacionais, podem ser utilizadas para SGBDs que possuam outros modelos de dados que não o relacional.

[10] Análise léxica e sintática

execução [SELI79]. Cada comando SQL é enviado para o *parser*, onde se verifica a correção da sintaxe. Um *bloco de requisição* é formado por uma lista SELECT, uma lista FROM e uma árvore WHERE, contendo, respectivamente, uma lista dos itens a serem recuperados, a(s) tabela(s) referenciada(s), e uma combinação booleana de predicados simples que são especificados pelo usuário. Um único comando SQL pode possuir muitos blocos de requisições, porque um predicado pode possuir um operando que pode ser outra requisição.

Se o *parser* não retorna erro de sintaxe, o módulo *otimizador* é chamado. O *otimizador*, por sua vez, acumula o nome das tabelas e colunas referenciadas em uma transação e verifica sua existência no catálogo do sistema, para obter informações a respeito de cada uma.

A porção do catálogo utilizada pelo otimizador também contém estatísticas sobre as relações referenciadas e sobre caminhos de acesso disponíveis para cada uma delas. Estas informações serão utilizadas posteriormente na *escolha do caminho de acesso*. Após a obtenção, no catálogo, do tipo de dado e o comprimento de cada coluna, o otimizador verifica novamente a lista SELECT e a árvore WHERE para verificar erros de semântica e incompatibilidade de tipos de dados.

Finalmente, o otimizador faz a escolha dos caminhos de acesso. Ele primeiramente avalia a ordem entre os blocos de requisição no comando. Então, para cada bloco, as relações na lista FROM são processadas. Se existir mais de uma relação em um bloco, são avaliados os métodos de *joins* disponíveis, levando-se em consideração permutações na ordem do *join*. O caminho de acesso que minimize o custo total de execução de um bloco é então escolhido a partir de uma árvore de alternativas de escolhas de caminhos de acesso.

Depois que um *plano de acesso*[11] é escolhido para cada bloco de requisição, o gerador de código é chamado finalmente. O gerador de código obtém, a partir da representação do plano de acesso escolhido, um código de máquina apropriado. Depois de gerado o código de máquina, este pode ser executado imediatamente ou então pode ser armazenado para execução posterior, dependendo da origem do comando, se foi de um programa ou de um terminal. Usualmente se denomina uma transação pré-compilada aquela originária de um programa e já armazenada no sistema na forma de código de máquina.

2.6.3 Plano de acesso

As propriedades quantitativas que sumarizam as instâncias de um banco de dados são seus *perfis estatísticos*. Eles descrevem aspectos tais como número de tuplas, número de valores distintos dos atributos, distribuição dos valores, correlação entre valores de atributos e a distribuição das tuplas nas unidades de armazenamento secundário. A estimativa de perfis é um elemento integrante do módulo de otimização de transações de um banco de dados e, em alguns casos,

[11] Veja a descrição mais detalhada de plano de acesso na seção 2.6.3 a seguir

também pode influenciar o projeto físico de banco de dados no problema de *seleção de índices*.

O objetivo principal de um *otimizador de transações* é derivar um *plano de acesso* eficiente para obter as informações desejadas pelo usuário. Um *plano de acesso* é uma descrição em alto nível de um programa que descreve o algoritmo, estruturas de arquivos, ordem das operações e aninhamento de *"loops"* internos e externos.

Para encontrar um plano eficiente, um otimizador gera e avalia um conjunto de alternativas. A avaliação das alternativas é baseada em fórmulas de custo que estimam o número de acessos à memória secundária, no caso de sistemas centralizados, ou o custo de comunicação e tempos de atraso para sistemas de arquitetura distribuída. Desde que estas fórmulas dependam direta ou indiretamente do tamanho de seus operandos, a estimativa de perfis estatísticos assume uma importância crítica no processo de otimização [MANN88]. Também, estas estimativas têm importância no projeto físico de banco de dados [FINK88]. Por exemplo, a escolha de índices é bastante influenciada pelo perfil estatístico dos atributos presentes nas cláusulas de operações de joins.

Outras soluções para a otimização de transações baseiam-se em promover um acoplamento entre um sistema dedutivo e o SGBD propriamente dito, mantendo-se as características de cada um ("acoplamento fraco"). De acordo com essa arquitetura, o sistema dedutivo representa o otimizador que, ao receber um consulta, gera uma consulta otimizada equivalente, que é passada ao SGBD. Neste otimizador, a geração dos planos de acesso é baseada em um modelo de otimização, cujas heurísticas são armazenadas como regras em uma base de conhecimento, ao contrário dos otimizadores convencionais, onde as heurísticas são incorporadas no próprio código [SILV90].

2.6.4 Estimativas de custos

O princípio econômico da otimização requer a maximização do *"output"* para uma dada configuração de recursos ou então, falando de outra forma, a minimização da utilização de recursos para a manutenção de um dado nível de *"output"* desejado. No caso de otimização de transações, o objetivo é minimizar os recursos necessários para avaliar uma expressão que recupera ou atualiza um banco de dados. Como recursos, podem ser considerados o tempo de resposta ou o esforço computacional. Apenas em sistemas de banco de dados distribuídos estes objetivos podem não ser coincidentes e o problema, geralmente, é muito mais complexo.

Os fatores que identificam o esforço computacional podem ser: tempo de serviço de CPU, tempo de espera por CPU (sistemas multiusuários), tempo de E/S [12], tempo de espera por E/S; em casos de sistemas distribuídos, o tempo de atraso de comunicação deve, também, ser utilizado [MANN88]. O custo de

[12] Operações de entrada e saída.

E/S é freqüentemente referenciado pelo número de acessos (leituras, escritas) lógicos a páginas. Uma *referência* a uma página do banco de dados é uma *referência lógica* se a página referenciada estiver presente no *buffer-pool*, caso contrário a referência se transforma em *referência física*. Para estimar o número de referências físicas, que são as que realmente degradam o sistema, aumentando o custo, devem-se considerar políticas de gerenciamento de *buffer-pool*. Para maiores informações sobre este tópico, ver [EFFE84] onde é apresentado uma implementação de um gerenciador de *buffer-pool*.

Em geral, otimizadores de transações utilizam hipóteses simplificadoras em seus modelos. Estas hipóteses são utilizadas não somente no campo específico de otimizadores de transações assim como em outras áreas de estudo de banco de dados. Em [CHRI84] são identificadas cinco hipóteses simplificadoras:

1. *Uniformidade de valores dos atributos*: estabelece que a probabilidade de ocorrência de um determinado valor de um atributo é a mesma para qualquer valor.

2. *Independência de valores dos atributos*: para 2 atributos dados, digamos A e B, significa que não existe nenhuma correlação entre valores de A com B.

3. *Uniformidade das transações*: as transações referenciam todos os valores dos atributos com a mesma freqüência.

4. *Número constante de tuplas por página*: é uma medida de uniformidade onde cada página do banco de dados possui um número médio de tuplas, ou seja, a probabilidade de uma página ser referenciada é $1/P$ onde P é o número de páginas.

5. *Disposição randômica das tuplas nas páginas* : estabelece que a forma como as tuplas são colocadas nas páginas não afeta a probabilidade de serem referenciadas, ou seja, a probabilidade de uma tupla ser referenciada é $1/N$ onde N é o número total de tuplas da relação.

Estas hipóteses simplificam a obtenção do custo de processamento de cada alternativa, mas, no entanto, podem também diminuir a precisão, afetando significativamente a escolha do melhor plano de acesso pelos otimizadores de transações. Com escolhas de planos de acesso ruins o desempenho final pode ser muito abaixo do esperado, pois, quando a distribuição de valores dos atributos segue uma distribuição muito assimétrica, os otimizadores de transações nos sistemas relacionais correntes freqüentemente falham na escolha do plano de acesso ótimo. Isso ocorre mesmo em se tratando de casos envolvendo transações simples que utilizam apenas uma relação. A causa principal desta falha reside na estimativa incorreta da seletividade. Em Lynch [LYNC88], é proposta uma série de novos métodos para se estimar a seletividade, que apresentam uma boa resposta para distribuições muito assimétricas.

Para esclarecer melhor a questão da utilização de hipóteses simplificadoras na obtenção de planos de acesso utilizaremos, como ilustração, um exemplo extraido de [MANN88]. Este exemplo se baseia em uma população de estudantes de uma universidade que são matriculados em vários cursos de especialização, como mostrado na Tabela 2.3. Podemos considerar que existem índices não agrupados [13] para os atributos IDADE e CURSO.

Para a seleção desejada, **listar todos os estudantes com mais de 33 anos e que também cursem "administração"**, existem 4 possíveis soluções. **Primeiro**, pode-se percorrer seqüencialmente toda a relação ESTUDANTE e verificar para cada tupla se as condições desejadas são satisfeitas; **segundo**, pode-se utilizar um índice em CURSO e acessar todas as tuplas de estudantes que possuam o curso de administracão e verificar para cada tupla escolhida se idade é superior a 33 anos; **terceiro**, pode-se acessar todos os estudantes com idade superior a 33 anos através do índice IDADE e verificar para as tuplas selecionadas aquelas que curso é igual a administração; **quarto**, pode-se obter através do índice IDADE um conjunto de tuplas com idade superior a 33 anos e através do índice CURSO todas as tuplas com curso igual a administração e obter a resposta fazendo-se a interseção entre os conjuntos.

As estimativas de custo podem ser obtidas de 3 modos diferentes:

1. *Valores reais*: utilizando-se os tamanhos reais, verificados para a população de dados colhida.

2. *Valores estimados por intervalo*: utiliza-se uma estimativa que leva em consideração o histograma e em cada classe de valores (intervalos), assume uma distribuição uniforme.

3. *Valores estimados*: os tamanhos são estimados tomando-se considerações de uniformidade e independência de dados.

Para o caso 1, podemos assumir que o número de tuplas que estão realmente qualificadas é 380 (o valor realmente verificado em um levantamento). Assumindo o modo de cálculo 2, obtemos 490 tuplas. Ao passo que, para uniformidade e independência de valores, cálculo 3, o valor estimado é 3000! [14]

2.6.5 Escolha de caminhos de acesso

Durante a fase de projeto físico, o projetista deve escolher quais índices irão ser construídos de modo a obter um melhor desempenho para o seu conjunto de transações. Tal problema é tratado como *escolha de índices* e em muito se assemelha ao problema da determinação do plano de acesso, pelo otimizador de transações.

[13] Aqueles em que a ordem dos índices não está relacionada com a ordem das tuplas nas páginas de dados.

[14] 3000 = 40.000 x 1/8 x 27/45

Tipicamente, um índice é implementado utilizando-se uma forma variante de *B-tree* [COME79], onde a *chave* de um índice é um conjunto de campos (atributos) de uma tabela na qual o índice é definido. Uma *chave simples* é constituída de somente um campo da tabela relacionada e uma *chave composta* é construída através da concatenação de múltiplos campos da tabela. Cada entrada em uma página no *nível folha* do índice consiste de um valor para a chave seguido de um ou vários RID, onde RID é o identificador do registro (*record indentifier*) que contém o valor da chave. Um dos índices de uma tabela pode ser designado como o índice de *agrupamento* - i.e. este índice é consultado durante uma operação de inclusão na tabela para obter uma página candidata para realizar a inclusão; a página candidata é a página que contém outros registros que possuem o mesmo valor na chave ou na mesma vizinhança do valor da chave.

Um índice é considerado um *índice elegível* para uso na resposta de uma transação, tanto à nível da escolha do usuário assim como na determinação do plano de acesso, se a chave contiver ao menos um dos campos referenciados nos predicados da transação.

Quando da *escolha dos índices*, o projetista de uma aplicação deve balancear, por um lado, as facilidades do acesso aos dados proporcionado pelos índices e, por outro lado, o custo correspondente de manutenção e armazenamento. Esta escolha, no entanto, não é nada trivial porque o número de possíveis escolhas pode ser muito grande. Se, por exemplo, indexássemos todas as colunas, dificilmente seria uma boa solução, porque independentemente da ocupação de espaço extra para armazenamento dos índices, os custos de atualização seriam muito grandes. Outra solução seria testar todas as combinações possíveis de escolhas de índices e escolher a melhor (pesquisa exaustiva); no entanto, esta solução pode gerar facilmente um número muito grande de combinações tornando seu uso inviável [MELO89].

O problema de escolha de índices é tratado em [FINK88], onde se descreve uma ferramenta para auxílio à determinação de soluções eficientes. Para uma carga de trabalho, caracterizada pelo conjunto de transações e suas respectivas freqüências de execução, a ferramenta sugere uma configuração física que conduz a um desempenho eficiente. Nesse modelo discute-se a forte integração entre o modelo de avaliação estatístico da ferramenta e aquele utilizado internamente pelo otimizador de transações do banco de dados. A ferramenta é implementada para o Sistema R [ASTR76] da IBM e as estimativas de custos e outras informações necessárias são obtidas de banco de dados utilizando o comando "EXPLAIN" da SQL. Em [MELO89] é apresentado um modelo analítico para solução da escolha de índices e colunas de *"agrupamento"*. O modelo utiliza hipóteses simplificadoras de uniformidade e independência de atributos. Em [FINK88], ainda argumenta-se que modelos externos, que não compartilham o mesmo modelo estatístico de análise utilizado pelo SGBD, podem apresentar alguns problemas tais como: a ferramenta pode tornar-se obsoleta sempre que ocorrer uma mudança no modelo do otimizador, já que mudanças são sempre constantes com o desenvolvimento de novas versões de um SGBD e, mais ainda,

o otimizador pode fazer suposições diferentes - e, em conseqüência, obter diferentes planos de acesso - daquelas obtidas pela ferramenta que utiliza um modelo externo. Assim, não teria o menor efeito a construção de índices por sugestão da ferramenta, se estes não fossem aproveitados pela escolha de plano de acesso, obtida pelo otimizador de transações.

Outro ponto importante em relação à escolha de índices, se refere a sistemas que possuem vários índices para uma mesma tabela, e, em particular, quando as tabelas possuem chaves compostas. Neste ponto, pouco se tem feito com relação a quantos índices utilizar, no caso da otimização de transações que impliquem na utilização de índices múltiplos [MOHA89].

Muitos SGBDs utilizam otimizadores que usam no máximo um índice, mesmo que existam disponíveis vários índices para a tabela desejada. Em [MOHA89] se utiliza uma técnica de *intersecção de índices* e uma técnica de *união de índices* que leva em consideração a existência de índices múltiplos para acessos a uma única tabela. Também, a ferramenta ABD/R [MELO88], considera mais de um índice por tabela.

Sendo assim, em sistemas que utilizam ao menos um índice para acesso a uma tabela, os usuários gostariam que a escolha do otimizador levasse em consideração a existência de vários índices, verificando os predicados que envolvessem várias colunas. No entanto, em muitos casos são forçados a definir índices em chaves compostas.

A definição de chaves compostas, por outro lado, é um problema de projeto de banco de dados não trivial, pois, além de requerer a decisão de quais campos serão incluídos, deve também decidir qual a melhor ordem. A partir do momento em que os SGBDs começassem a utilizar técnicas que considerassem o uso de índices múltiplos, então haveria menos motivação para a definição de índices sobre chaves compostas, o que facilitaria o problema de projeto de bancos de dados.

2.7 Carga de trabalho

A *carga de trabalho* é composta por um conjunto de transações e de suas respectivas freqüências de execução. Muitas vezes se denomina *carga de trabalho do banco de dados* (*"database workload"*) aquela proveniente apenas das transações que são submetidas pelos usuários, e *carga de trabalho externa* (*"NonDBMS workload"*) aquela provocada pelos processos que são executados concorrentemente ao SGBD e que disputam os mesmos recursos computacionais: unidades de disco, processador e memória principal.

A carga de trabalho do SGBD é caracterizada pela estrutura do banco de dados, o conjunto de transações que são executadas e o número de usuários utilizando simultâneamente o sistema [TURB87]. Em sistemas distribuídos, a sobrecarga provocada pelo controle da execução das transações e a troca de informações entre os nós da rede, também deve ser considerada [STON82].

Entre os parâmetros mais relevantes para a descrição das transações incluem-se:

- *Tipo de transação*: seleção, projeção, inserção, remoção, atualização, junção, agregação ou, ainda, transações mais complexas que envolvam mais de um tipo.
- *Predicados*: número, tipo e independência de predicados.
- *Valores finais e intermediários*: tamanho de tuplas e atributos como resultado das transações.
- *Freqüência relativa das transações*: para o conjunto das transações encontradas nas aplicações, organizadas por tipo de transação.

Em vários modelos teóricos empregam-se *hipóteses simplificadoras* que assumem uniformidade no comportamento das transações de modo a facilitar a utilização desses modelos. Em geral, assume-se que todas as transações referenciam os valores dos atributos com a mesma freqüência [CHRI84].

No entanto, tem sido demonstrado, através de estudos experimentais, que o comportamento da carga de trabalho é não uniforme. Geralmente, um conjunto relativamente pequeno, e freqüentemente utilizado, de transações e' responsável pelo maior consumo de recursos, de acordo com a regra 80-20 [MAGA81].

Em [YU89], pesquisou-se o comportamento da carga de trabalho em banco de dados relacionais e concluiu-se que aproximadamente 90% das transações encontradas eram somente transações de consulta (*"read-only"*). Além disso, pode-se verificar que, grande parte das transações consumiam poucos recursos e um pequeno número de transações provocavam um grande consumo de recursos (87% das transações utilizando apenas um índice examinavam 10 tuplas para encontrar a tupla qualificada, até 99.6% examinavam até 100 tuplas, e o restante, 0.4% das transações, provocavam buscas extremamente longas), evidenciando o comportamento não uniforme.

Outros estudos de importância, procuraram avaliar as interações entre as cargas de trabalho do banco de dados e as cargas de trabalho externas, procurando identificar quais tipos de carga de trabalho mais interferem no funcionamento dos SGBDs. Em [TURB87] foram pesquisados combinações envolvendo 3 tipos de processos, o primeiro era caracterizado por uma alta taxa de operações de E/S ("I/O bound"), o segundo dava ênfase à utilização de CPU ("CPU bound") através de processamento numérico apenas, o terceiro estabelecia artificialmente diferentes níveis de freqüências de geração de operações de *"page-fault"*[15] para se testar o desempenho do gerenciamento de memória.

[15] Quando uma referência lógica não encontra a página desejada no *"buffer"*.

Tabela 2.1: Comparação entre modelos de análise de desempenho

modelo	aderência	*workload*	conteúdo banco de dados	controlabilidade	generalidade
analítico	−	−	−	+	+
simulação	+−	−	−	+	+
workload fictícia	+	+−	+	+	−
workload real	+	+−	+	+	−

Tabela 2.2: Distribuições do Tipo-Zipf

Rank	Freqüências			
	$z = 0$	$z = 0.5$	$z = 1$	$z = 3$
1	0.1	0.1992	0.34	0.8351
2	0.1	0.1408	0.17	0.1044
3	0.1	0.1150	0.11	0.0309
4	0.1	0.0996	0.09	0.0130
5	0.1	0.0891	0.07	0.0067
6	0.1	0.0813	0.06	0.0038
7	0.1	0.0753	0.05	0.0024
8	0.1	0.0704	0.04	0.0016
9	0.1	0.0664	0.04	0.0012
10	0.1	0.0629	0.03	0.0009
TOTAL	1.0	1.0000	1.00	1.0000

Tabela 2.3: Número de alunos x faixa etária x curso

Curso	Idade						
	16-20	21-25	26-30	31-35	36-40	41-60	Total
Economia	2.045	3.600	3.625	400	175	155	10.000
Pedagogia	275	500	750	625	250	100	2.500
Engenharia	750	1.800	1.775	450	125	100	5.000
Artes	2.250	3.000	3.600	500	400	250	1.000
Administração	250	575	875	425	250	125	2.500
Biologia	775	1.850	2.000	150	150	75	5.000
Enfermagem	225	550	375	200	120	30	1.500
Sociologia	575	900	1.200	500	225	100	3.500
Total	7145	12775	14200	3250	1695	935	40.000

Capítulo 3

Benchmark de banco de dados

Nos últimos anos, presenciamos uma grande proliferação de sistemas de bancos de dados disponíveis comercialmente. Tais sistemas apresentam avanços em termos de incorporação de novas propostas de modelos de dados assim como novas arquiteturas de sistemas, com o oferecimento, inclusive, de *hardwares* especializados para funções de bancos de dados.

Se considerarmos somente os sistemas de tecnologia relacional, já existem atualmente vários deles que exibem um desempenho considerável. Além disso, podem ser encontrados sistemas disponíveis em equipamentos que vão desde os computadores pessoais e *workstations* até os sistemas de grande porte. Todos esses sistemas, apesar de poderem suportar a construção de sistemas relacionais capazer de obter um bom desempenho para recuperações e atualizações de informações, diferem enormemente tanto em custo como em capacidade.

Devido a essa grande variedade de sistemas disponíveis, torna-se imprescindível uma melhor avaliação da relação custo x benefício. A necessidade de estudos sobre o comportamento dos sistemas tem levado ao desenvolvimento e aperfeiçoamento de várias técnicas de análise de desempenho. Entre elas, a técnica de *benchmark de bancos de dados* vem sendo uma das mais utilizadas, tendo sido adotada, inclusive, como padrão de avaliação de desempenho de SGBDs [ANON85], [BITT83].

A técnica de *Benchmark de Banco de Dados* consiste na execução de um conjunto de transações [1] sobre um banco de dados conhecido, com o objetivo de medir o desempenho do sistema. Trata-se de uma técnica experimental onde os testes são realizados sobre o próprio sistema de banco de dados composto pelo SGBD e o ambiente computacional (*software* e *hardware*) onde o sistema está instalado.

[1] Geralmente denominada carga de trabalho do SGBD (*database workload*).

A utilização da técnica de *benchmark*, por se tratar de uma técnica experimental, requer que o sistema a ser medido deva estar pronto e disponível para uso. As técnicas teóricas e de simulação, ao contrário, podem ser utilizadas em sistemas que se encontram em fase de concepção e não haveria possibilidade de testá-los. Dentre as vantagens da técnica de *benchmark* sobre outras técnicas de análise de desempenho destacam-se a fidelidade dos resultados obtidos e a relativa facilidade de execução.

Na técnica de *benchmark* de bancos de dados podem ser utilizadas cargas de trabalho e banco de dados, tanto *sintéticas* quanto *reais*. O termo sintético, aqui empregado, refere-se a um conjunto de transações e de dados que procuram exercitar o sistema consumindo recursos, mas que, em contrapartida, não realizam nenhum trabalho útil (ou real). A qualificação de real para a carga de trabalho e para o banco de dados refere-se aos programas e aos dados que são utilizados em um sistema em produção.

A seguir, apresentamos um resumo dos trabalhos mais recentes publicados na área de *benchmark* de banco de dados, destacando a metodologia apresentada no *benchmark de Wisconsin* [BITT83] e a metodologia do *benchmark de Débito e Crédito* [ANON85]. Tais metodologias vêm sendo largamente utilizadas para análise de desempenho de SGBDs, tanto comerciais como acadêmicos, constituindo-se, na prática, em verdadeiros padrões de análise de desempenho.

3.1 Aplicações da técnica de *benchmark*

A técnica de *benchmark de sistemas* tem sido freqüentemente utilizada em trabalhos sobre *análise de desempenho*. Ela pode ser utilizada não somente na área de banco de dados especificamente, como, também, em outras áreas da Ciência da Computação, de uma maneira geral. No caso de *benchmarks de bancos de dados*, um conjunto conhecido de transações é executado sobre o banco de dados (fictício ou real) instaladas no próprio sistema que se deseja avaliar.

Ao contrário de outras técnicas de análise de desempenho, a técnica de *benchmark* requer que o sistema a ser testado esteja pronto e disponível para uso. Já os modelos analíticos (teóricos) e de simulação podem, também, ser utilizados em fases de concepção do sistema, ou durante a fase de projeto, quando não se dispõe ainda de dados operacionais.

De um ponto de vista prático, a técnica de *benchmark* oferece algumas vantagens sobre outras técnicas de análise de desempenho, no que se refere à fidelidade dos resultados. Devido ao fato dos testes serem realizados sobre o próprio sistema, pode-se ter uma confiança maior nos resultados obtidos com *benchmark* do que com modelos teóricos ou de simulação, por exemplo.

A *técnica de monitoramento* pode ser considerada como a técnica que tem a capacidade de produzir os resultados mais fidedígnos. No entanto, sua aplicação é também considerada a mais onerosa. Não raramente, as ferramentas contruídas para se fazer monitoramento de sistemas tornam-se muito específicas

para o sistema que se deseja analisar, prejudicando o seu transporte para outros sistemas. Mesmo as ferramentas de monitoramento que são oferecidas nos próprios SGBDs, não possuem, em geral, um padrão uniforme de medição, o que torna impossível a comparação entre os dados obtidos de sistemas diferentes [BORA84].

A aplicação da técnica de *benchmark* pode ser considerada pouco onerosa do ponto de vista do seu desenvolvimento e aplicação, comparando-se com a técnica de monitoramento. Além disso, as medidas utilizadas são, em geral, medidas macroscópicas, tais como o tempo de resposta e *tps*, que também são mais fáceis de serem obtidas.

O uso de *benchmarks* de bancos de dados, por outro lado, não está restrito apenas à análise de desempenho. Em muitas situações pode-se utilizar a técnica de *benchmark* para testes de verificação de correção de implementação e de conformidade com determinados modelos de SGBDs. Nestes casos, devido à grande complexidade dos sistemas a serem testados, como é o caso dos SGBDs, a verificação de correção através de comprovações teóricas ainda é inviável. Porém, a aplicação de *benchmark* é mais freqüente para análise de desempenho e sua utilização em testes de correção de programas ocorre mais comumente durante os estágios iniciais de implementação de um SGBD, tal como ocorreu nas primeiras implementações de SGBDs utilizando o modelo relacional, INGRES e SYSTEM R [TURB87].

Em resumo, a utilização de *benchmarks* é mais popular como ferramenta de análise de desempenho. Entre as muitas das situações típicas em que é utilizada, uma das mais comuns é a escolha de um SGBD.

Uma comparação entre SGBDs, utilizando-se *benchmark*, será correta desde que o padrão aplicado em cada um dos sistemas analisados seja o mesmo. Caso o *benchmark* seja realizado utilizando-se uma aplicação existente na empresa interessada em comparar os possíveis sistemas, então, essa mesma aplicação deve ser codificada para cada um dos sistemas a serem testados. Este esforço é, sem dúvida, muito dispendioso consirerando-se que se deseja apenas escolher um sistema. Neste caso, o usuário pode propor um *benchmark sintético*, um pouco mais simplificado, que pode ser mais fácilmente aplicado. A definição do *benchmark* deve preferencialmente procurar refletir algumas das características da aplicação do usuário, conferindo aos testes uma maior representatividade.

Outro fator que deve ser levado em consideração, refere-se à execução do *benchmark*, que pode estar tanto ao encargo do usuário ou então dos próprios fornecedores interessados na venda dos seus sistemas. No caso do próprio usuário executar o *benchmark*, ele adquire uma chance única de conhecer não somente os resultados de desempenho do sistema, mas, também, outros aspectos relativos à qualidade do produto em si, tais como: documentação, suporte, facilidade de uso etc. Em [STON85] sugere-se que, sempre que possível, os próprios usuários deveriam planejar e executar o *benchmark*, ao invés de apenas receber e analisar os resultados finais apresentados pelos fornecedores; porque além das vantagens já mencionadas anteriormente, pode-se evitar que os próprios fornecedores

alterem o *benchmark* de modo a melhor se adaptar à seus produtos.

Atualmente, já existem alguns *benchmarks* bastante populares [BITT83], [ANON85] e, apesar das muitas críticas e polêmicas por eles levantados, continuam a ser desenvolvidos e vêm sendo extensamente utilizados. Como fruto destes trabalhos, uma quantidade razoável de sistemas têm sido testados e, cujos resultados encontram-se disponíveis para os usuários através de várias fontes: publicações em revistas especializadas, apresentações em simpósios e discussões em encontros de especialistas da área.

Entretanto, apesar da facilidade proporcionada por estas avaliações e a disponibilidade de dados, deve-se levar em consideração que estes resultados são válidos somente nas condições em que foram obtidos. Ou seja, os resultados conseguidos através de *benchmarks* estão de alguma forma restritos no tempo, visto que eles avaliam apenas uma determinada versão de um SGBD. Sendo assim, não se pode fazer considerações genéricas e extrapolações de resultados sob pena de cometerem-se sérios enganos, pois, a cada dia, novas versões tornam-se disponíveis, apresentando resultados significativos em comparação com as anteriores.

Em geral, as críticas feitas às metodologias de *benchmark* abordam aspectos controvertidos que são difíceis de se tratar na prática tais como: a formação do banco de dados, o seu conteúdo e disposição dos dados, e a carga de trabalho utilizada. Em muitos casos, as críticas se justificam porque consideram que as características de alguns *benchmarks* não são suficientemente representativas e genéricas para se avaliar qualquer sistema [HAWT85].

Entre as várias metodologias de *benchmarks* propostas na literatura, duas delas têm uma importância especial devido à grande popularidade que alcançaram: a metodologia de Wisconsin, mais conhecida por *benchmark de Wisconsin* ou *benchmark de DeWitt* [BITT83], [DEWI85] e o *benchmark de Débito e Crédito* [ANON85]. A primeira pode ser considerada praticamente como um padrão para análise de SGBDs relacionais e a segunda como um padrão de sistemas de transações *on-line* (OLTP[2]).

3.2 Benchmark de Wisconsin

O *benchmark de Wisconsin* (**BW**) pode ser considerado atualmente como um padrão para análise de desempenho de SGBDs relacionais. Vários sistemas, tanto comerciais quanto acadêmicos, já foram avaliados com este *benchmark*, tais como: INGRES, ORACLE, IDM500, SQL/DS, entre outros. O **BW** foi uma das primeiras tentativas de formalizar a avaliação de desempenho em sistemas relacionais e devido à sua simplicidade, portabilidade e o pouco tempo necessário para sua execução, fez com que fosse largamente adotado por indústrias e laboratórios de pesquisa. Como resultado, medidas de desempenho de uma variedade de sistemas, com diferentes configurações, tornaram-se disponíveis

[2] *On-line Transaction Processing*

para usuários que desejassem comparar o desempenho dos vários sistemas já testados.

As caracteristícas gerais do *benchmark* de Wisconsin são a utilização de um banco de dados e um conjunto de transações sintéticas (ou fictícias), ambiente de execução *mono-usuário* e *"stand-alone"*.

No ambiente *mono-usuário*, as transações do *benchmark* são submetidas seqüencialmente, como se apenas um usuário as estivesse executando, internamente ao SGBD. A característica *"stand-alone"* significa que não existem outros processos, externos ao SGBD, sendo executados concorrentemente, no mesmo sistema computacional.

As características mono-usuário e *stand-alone* visam, em primeiro lugar, facilitar a aplicação da metodologia. A utilização de dados fictícios permite sua própria obtenção através de programas específicos, facilitando a portabilidade em diversos sistemas, permitindo, assim, uma melhor padronização.

A configuração de um ambiente mono-usuário, por um lado, deixa de testar várias características que são importantes em sistemas reais tais como: concorrência, implementação de *locks*[3], algoritmos de *read-ahead* e de *write-behind* etc.; por outro lado, torna a execução dos testes mais simples e também menos onerosa. Os resultados da execução de uma transação neste ambiente têm o significado de medir o desempenho do sistema para um condição ótima, em que não existe concorrência [DEWI85], [BITT89]; ou seja, o tempo de execução de uma transação neste ambiente deve ser considerado como o melhor tempo de resposta possível, para o sistema que está sendo testado.

Um banco de dados sintético, de teste, proporciona a possibilidade de ser gerado através de programas específicos. Sua estrutura, bastante simples, facilita o entendimento e mesmo um usuário, que não conheça-o préviamente, com pouco tempo de aprendizado aprende a utilizá-lo e facilmente realiza consultas com total controle sobre os resultados obtidos. As relações que compõem o banco de dados são bastante flexíveis e permitem consultas com vários graus de seletividade.

O principal objetivo do *benchmark* de Wisconsin é procurar obter dados de desempenho do SGBD que sejam independentes da escolha de uma aplicação (*benchmark de sistema*). Neste sentido, são realizados testes sobre blocos básicos de implementação, tais como: soluções de algoritmos de operações de *"joins"*, mecanismos de indexação, escolha de plano de acesso etc.

Com este propósito em mente, o controle proporcionado pelo banco de dados sintético é muito maior que aquele sobre um banco de dados real. Para se conhecer a distribuição dos dados em um banco de dados real é necessário realizar muitos testes, e, em muitos casos, o nível de controle que se deseja obter nem sempre é possível.

[3] *Locks* são o principal mecanismo de controle de concorrência utilizado para garantir a consistência de dados durante leituras e atualizações em um ambiente de vários usuários simultâneos.

Tabela 3.1: Descrição dos atributos das relações do **BW**

Nome	Tipo	Intervalo	Ordem	Comentário
unique1	int	0 - 9999	randômico	chave primária
unique2	int	0 - 9999	randômico	
two	int	0 - 1	cíclico	0,1,0,1,...
four	int	0 - 3	cíclico	0,1,2,3,0,1,...
ten	int	0 - 9	cíclico	0,1,...9,0,1,...
twenty	int	0 - 19	cíclico	0,1,...19,0,1,...
hundred	int	0 - 99	cíclico	0,1,...99,0,1,...
thousand	int	0 - 999	randômico	0,1,...999,0,1,...
twothous	int	0 - 1999	randômico	0,1,...1999,0,1,...
fivethous	int	0 - 4999	randômico	
odd100	int	50	cíclico	1,3,5,...,99,1...
even100	int	50	cíclico	2,4,...,100,2,..
stringu1	char	A..A..A-V..V..T	randômico	chave primária
stringu2	char	A..A..A-V..V..T	cíclico	chave primária
string4	char	A..A..A-V..V..V	cíclico	

Em um banco de dados sintético, se o usuário desejasse fazer um teste onde é preciso ter o controle da seletividade e, digamos, construir uma transação que possua uma seletividade de 10%, isso é perfeitamente possível utilizando-se as relações de Wisconsin, como veremos adiante. Já para um banco de dados real, nem sempre este valor exato pode ser obtido; quando muito, pode-se obter um valor aproximado. Entretanto, dificilmente a seletividade pode ser controlada, fazendo-se testes para qualquer outro valor que venha a ser escolhido.

Outro fator importante com relação ao banco de dados sintético encontra-se na sua portabilidade; pois, sendo ela gerada através de programas, torna-se mais fácil carregá-la para qualquer sistema e conseqüentemente o estabelecimento de um padrão de análise de desempenho é facilitado, o que é muito importante.

O banco de dados de Wisconsin é constituido de 4 relações, *OneKtup, TwoKtup, FiveKtup e TenKtup* , com 1000, 2000, 5000 e 10000 tuplas respectivamente. Cada relação possui 16 atributos, sendo 13 atributos de tipo inteiro de tamanho igual a 2 bytes, e 3 atributos do tipo *string* [4]. Todas as relações possuem os mesmos atributos, apenas modificando-se a cardinalidade em cada relação. Na tabela 3.1 mostram-se as construções de cada atributo (domínios). Os valores dos atributos das relações são distribuídos uniformemente, permitindo a obtenção de vários graus de seletividade. Os intervalos de valores de atributos do tipo inteiro são indicados no próprio nome, assim, *two* possui um intervalo

[4] Cadeia de caracteres

Tabela 3.2: Fragmento da Relação OneKtup

Unique 1	Unique 2	Two	Ten	Hundred	Thousand
378	0	1	3	13	615
616	1	1	4	4	695
910	2	0	6	26	313
180	3	0	2	52	74
879	4	0	0	20	447
557	5	1	9	29	847
916	6	0	7	47	247
73	7	1	4	54	455

de 0 a 1, *four* de 0..3, ..., *hundred* de 0..99, *thousand* de 0..999, assim por diante.

O atributo unique1 é chave primária das relações e assume valores únicos, aleatóriamente dispostos sobre a relação. Para *OneKtup* os valores de unique1 vão de 0 .. 999. Na tabela 3.2 mostra-se um fragmento da relação *OneKtup*. O atributo unique2 também assume valores únicos, mas sua disposição na relação é seqüencial. Para a realização do testes, são utilizadas relações sem e com agrupamento, tomando o atributo unique2 como referência. Os atributos tipo *string* (stringu1 e stringu2) possuem comprimento fixo de 52 bytes e valores únicos. São formados por caracteres significativos nas posições 1, 27 e 52 da cadeia sendo o restante composto por caracteres não significativos ("x"); desta forma é possível a formação de até 26^3 combinações de cadeias, o suficiente para preencher a relação *TenKtup*.

A mesma estrutura do banco de dados utilizado no **BW** foram utilizadas, posteriormente, em outras propostas de *benchmark* [BORA84], [TURB87]. Em [BITT89] se fornece um *script* completo (em SQL), contendo o esquema do banco de dados e o conjunto de transações utilizados no **BW**.

Devido à forma como são construídas as relações, mesmo um usuário, que não possua um conhecimento prévio sobre elas, pode facilmente entender sua estrutura e aprender a utilizá-las. Pode-se construir, com grande facilidade, várias transações com diferentes graus de seletividade. Por exemplo, uma transação em SQL, que recupera 10% das tuplas de uma tabela, poderia ser:

SELECT * FROM OneKtup WHERE unique1 < 100

Em outro exemplo, utilizando-se a tabela TwoKtup, poderia-se fazer uma projeção com 95% de valores repetidos.

SELECT DISTINCT(hundred) FROM TwoKtup

O conjunto de transações do **BW** é formado por um conjunto variado de transações que incluem transações simples, tal como uma recuperação de uma tupla em uma relação ou a busca em um determinado intervalo, agregações (*min* e *sum*), junções de 2 ou 3 relações, projeções, atualizações e inserções. Durante o *benchmark*, as mesmas tuplas que são inseridas, são, posteriormente, atualizadas e removidas. Ao final do *benchmark* todas as relações conservam as mesmas características iniciais, a mesma cardinalidade e distribuição de valores. Em [BITT83] e [BITT89] podem ser encontrados maiores detalhes sobre o conjunto de transações do **BW**.

No **BW** empregam-se apenas dois tipos de dados: numérico e *string*. Os valores numéricos são de apenas 2 bytes e os de tipo string são de 52 bytes com comprimento fixo. Estes parâmetros têm sido muito criticados e dificilmente poderiam ser justificáveis atualmente. No *benchmark* proposto por Turbyfill [TURB87] se utilizam atributos inteiros com 4 bytes e atributos tipo string com tamanho de 20 bytes, pois, de acordo com especialistas, os tamanhos variando de 20 a 30 bytes são mais freqüentes em aplicações reais. Inclusive é mais realista supor-se a utilização de *strings* com tamanho variável [TURB87], o que também pode ser mais interessante por introduzir um comportamento não uniforme no banco de dados (número de tuplas por página variável). Outro ponto de falha no **BW** é a ausência de operações envolvendo decimais e ponto flutuante, onde um dos efeitos mais importante de operações deste tipo, além da sobrecarga no armazenamento, é a sobrecarga de processamento.

Quanto ao fato dos testes serem realizados em ambiente mono-usuário, isso não invalida totalmente os resultados alcançados, pois, se uma transação não possui um bom desempenho neste ambiente, não faria o menor sentido testá-la em ambiente multi-usuário. Além disso, existe também a possibilidade de permitir a verificação da existência de determinadas anomalias no sistema em teste, principalmente devido ao uso de algoritmos de *joins* inadequados [DEWI85]. Quanto ao fato dos testes serem realizados em um ambiente *stand-alone*, a crítica que normalmente se faz é a de que os SGBDs rodam, na maioria dos casos, em computadores não dedicados. Sendo assim, torna-se muito importante verificar o efeito da interação do SGBD com os outros processos que estão sendo executados simultaneamente na mesma máquina [TURB87].

3.2.1 Variações do BW

Vários outros *benchmarks* foram propostos posteriormente, baseando-se na metodologia do **BW**. Logo no ano seguinte ao da publicação da primeira versão do **BW** [BITT83], propôs-se uma extensão na metodologia para ambientes multiusuários em [BORA84].

Na *metodologia de benchmark multiusuário*, os principais fatores analisados e considerados hipotéticamente como sendo os mais críticos em ambientes multiusuário foram:

1. Nível de Multiprogramação (NMP)
2. Grau de Compartilhamento de Dados (GCD)
3. Mistura de Transações (MT)

O *nível de multiprogramação* é definido como o número de processos que são executados simultaneamente pelo processador de transações. O controle do *grau de compartilhamento de dados* (medido em termos de porcentagens entre 0 a 100%) obteve-se duplicando as relações. Assim um GCD de 0% significa que não existe nenhum compartilhamento, ou seja, cada transação acessa exclusivamente uma relação, ou cópia da relação. Para um GCD de 100% significa que todas as transações acessam as mesmas relações. A *mistura de transações* obteve-se a partir de um critério de utilização de recursos, onde se selecionam transações que melhor representem os seguintes tipos de comportamento:

- ↓ CPU e ↓ Operações de E/S
- ↓ CPU e ↑ Operações de E/S
- ↑ CPU e ↓ Operações de E/S
- ↑ CPU e ↑ Operações de E/S

Nesta classificação, as setas significam um nível alto (seta para cima) e nível baixo (seta para baixo) de utilização de um determinado recurso (CPU ou operações de E/S).

A partir desta classificação entre quatro tipos de transações, formam-se vários tipos de misturas (composição de transações) para as quais se realizam vários testes, incluindo-se diferentes valores para os outros dois parâmetros, GCD e NMP. Também observou-se que a escolha das transações que melhor representam cada um dos quatro tipos de transação desejadas é dependente do sistema que se está analisando e, portanto, a determinação do conjunto de transações de teste deve ser realizada caso a caso [DEWI85].

Outros pesquisadores, insatisfeitos com os resultados obtidos com o *benchmark* multiusuário de Wisconsin [DEWI85], argumentam que o objetivo de se construir um *benchmark* independente de aplicação, i.e. genérico, é muito difícil de se atingir na prática. Tal *benchmark* precisaria ser representativo frente a interações muito complexas de fatores, tais como, carga de trabalho × algoritmos × hardware. Em uma análise do sistema IDM500 [HAWT85] foram feitas sugestões de algumas extensões ao *benchmark multiusuário*, baseando-se em um modelo de processamento mais genérico, que levasse em consideração um quarto fator: a quantidade de dados retornada ao usuário, além dos 3 fatores analisados inicialmente em [BORA84]. Argumenta-se que, se este fator não for levado em consideração, o aumento de desempenho utilizando-se o sistema IDM500 não poderia ser medido convenientemente.

O *benchmark* desenvolvido em [TURB87], chamado AS^3AP (*Ansi SQL Standard Scalable And Portable*), possui um enfoque ainda mais abrangente. O objetivo desejado é tornar o *benchmark* "portável" entre arquiteturas diferentes e "escalonável" para facilitar comparações entre sistemas de diferentes capacidades. Além disso, deve ter, também, a capacidade de fornecer informações úteis ao projetista de sistemas, permitindo-se a configuração de várias cargas de trabalho.

Além de teste básicos, o AS^3AP possui também um módulo específico de testes para o otimizador de transações. Nestes testes se utiliza um banco de dados com estruturas semelhantes a do *benchmark* de Wisconsin, mas que são alteradas utilizando-se atributos do tipo *string* com tamanhos variáveis e distribuições de valores assimétricas do tipo Zipf.

Para a avaliação de sistemas de diferentes arquiteturas, o AS^3AP procura fornecer a possibilidade de ser utilizado em uma escala compatível ao sistema que se deseja analisar. O fato de modificar o "tamanho" de um banco de dados de acordo com uma determinada arquitetura, proporciona uma oportunidade particular de comparação, a *razão de equivalência de bancos de dados*, que determina que, para uma comparação entre um SGBD X e um SGBD Y, a razão de equivalência seja calculada por:

(tamanho do SGBD AS^3AP em X) $\div$ (tamanho do SGBD AS^3AP em Y)

na qual, o sistema X e o sistema Y exibem *desempenhos equivalentes*. Neste caso, *desempenho equivalente* corresponde a uma distribuição de tempos de reposta para transações do AS^3AP equivalentes, onde nenhum dos sistemas comparados é sempre mais rápido ou sempre mais lento. Por exemplo, pode-se determinar que um banco de dados de 10 gigabytes em um SGBD distribuído possua uma escala equivalente a um banco de dados de 10 MBytes em um SGBD centralizado utilizando-se um único computador de uso geral. Com esta metodologia, a razão de equivalência passa a ser uma medida confiável quando a arquitetura, a capacidade de hardware e as versões de software são mantidas constantes, e, a partir deste ponto, seria correto utilizarem-se os custos financeiros dos sistemas como critérios de comparação [TURB87].

3.3 Benchmark de Débito-Crédito

O chamado *benchmark de Débito e Crédito* [ANON85] foi desenvolvido visando criar um padrão de transação para classificar o nível de desempenho de sistemas comerciais tipo *OLTP*. Devido às características adotadas, obteve bastante aceitação no ambiente da computação comercial.

Basicamente, o *benchmark de Débito e Crédito* consiste de 3 testes multiusuários, sendo o mais popular o teste denominado **TP1** (figura 3.1). Em síntese, o teste **TP1** consiste na execução de transações comumente encontradas em aplicações bancárias do tipo depósito e débito em uma conta corrente.

No **TP1**, como proposto originalmente em [ANON85], utiliza-se um terminal conectado ao equipamento principal via **X-25**. Inicialmente o sistema faz um serviço de apresentação para mapear a entrada para um programa COBOL, que utiliza um sistema de banco de dados para debitar ou creditar uma conta bancária. Depois, faz a contabilidade na forma de "partilhas-dobradas" e então responde ao terminal. Para realizar uma transação deste tipo, é requerido que noventa e cinco por cento (95%) das transações sejam executadas com tempo de resposta abaixo de 1 segundo.

Além do teste **TP1**, o *benchmark de Débito e Crédito* possui testes de busca e atualização seqüencial de registros, e testes de ordenação de um grande número de registros. Mais detalhes com relação a estes testes podem ser encontrados em [ANON85].

Um dos principais objetivos da metodologia adotada é obter um *padrão de transação* definida pelo tipo e limitada pelo tempo de resposta. No caso do teste TP1, mede-se o número de *tps* que são executadas pelo SGBD sendo que o tempo de resposta deve estar abaixo de 1 segundo.

Com a padronização destes testes, os sistemas comerciais poderiam informar os usuários e estes, por sua vez, especificar e escolher seus sistemas baseando-se em medidas de *tps* mais confiáveis.

Entre as principais críticas que se fazem ao *benchmark* de Débito e Crédito se colocam:

- *Não aderência a padrões estabelecidos*: a execução do *benchmark* obtém melhores resultados se não forem utilizadas somente características do padrão SQL. Entretanto, idealmente não se deveria construir um *benchmark* com o objetivo de tornar-se padrão, onde a melhor solução utiliza caracteríticas que não fazem parte da ANSI/SQL.

- *Representatividade*: o TP1 é um *benchmark* de uma operação de um cheque bancário (saque ou depósito) onde se assume uma propriedade invariante que é questionável: o sistema mantém constante a quantidade de dinheiro na gaveta de cada caixa, assim como a quantidade em cada agência do banco. Estas hipóteses podem ser facilmente criticadas e a solução adotada pode ser considerada muito simplista.

- *Baixa complexidade*: o TP1 não possui transações de consulta, não testa *joins* entre relações, otimização de transações, consultas aninhadas ou funções sobre agregados e nem tampouco inclui controles de lógica ou desvios. Ou seja, o TP1 não testa aplicações reais, é somente um teste de quão rapidamente um SGBD pode processar um grande número de atualizações [FINK90].

3.4 Outros *benchmarks*

Existem ainda outros *benchmarks* mais recentes, onde se procuram avaliar certos parâmetros de desempenho considerando-se tipos de ambiente menos convencionais.

Com o aumento da capacidade computacional, e o baixo custo de equipamentos que fornecem alto desempenho, as estações de trabalho (*workstations*) passaram a se tornar freqüentes em aplicações de projetos de engenharia do tipo CAD/CAM. A tecnologia de bancos de dados, por sua vez, também evoluiu no sentido de suportar mais eficientemente os requisitos das aplicações usualmente encontradas nestes tipos de ambiente.

Para atender a esta necessidade, desenvolveu-se a teoria dos *SGBDs orientados a objetos complexos*, que, apesar de não existir ainda nenhum modelo já padronizado, algumas das caracterísiticas normalmente associadas a estes sistema são: encapsulamento, herança, polimorfismo etc. Atualmente, já se encontram disponíveis comercialmente sistemas que atendem algumas destas características [ZDON90].

No chamado *benchmark SUN* [RUBE87], o interesse principal é analisar os parâmetros mais significativos para sistemas de banco de dados orientados para aplicações de engenharia e projeto (CAD) utilizando plataformas tipo estações de trabalho (*workstation*). Nesta abordagem o tempo de resposta é considerado como o ponto mais crítico e as operações efetuadas são menos complexas. Comparações utilizando este *benchmark* entre sistemas de banco de dados relacionais e sistemas orientados a objetos, podem ser encontrados em [DUHL88]. Em [ANDE89] é apresentado um *benchmark* para avaliação de SGBDs em ambientes de projeto e engenharia. Nessa metodologia se utiliza um paradigma baseado em hipertexto como meio de representação genérica de uma aplicação típica que é utilizada em ambientes como este.

No campo comercial, mais especificamente na área de bancos de dados transacionais, recentemente foi proposta uma padronização, em torno de um *benchmark* que se baseia na metodologia do *benchmark* de Débito e Crédito, por uma associação denominada *TPC* [5] [TPC89], formada pelos principais fabricantes de computadores de grande porte (AT&T, IBM, Bull, Hitachi, Unisys etc.), de estações de trabalho (SUN, Pyramid, HP) e fornecedores de SGBDs (Oracle, Informix, Software A.G., Relational Technologies, entre outros). O objetivo da *TPC* é desenvolver padrões de *benchmark* para a medição de sistemas OLTP.

O primeiro *benchmark* padronizado pelo *TPC*, chamado *TPC Benchmark A* (TPC-A), é definido apenas funcionalmente, da mesma forma que o TP1, do qual é derivado. Assim cada fornecedor de SGBD pode possuir uma definição própria do *benchmark*. O TPC-A é definido através de uma especificação em alto-nível que pode ser implementada em qualquer sistema operacional, utilizando qualquer linguagem de programação e plataforma de *hardware*. Além das

[5] Transaction Processing Performance Council

recomendações do *benchmark* propriamente, o TPC-A recomenda uma auditoria no SGBD e propõe uma lista de itens a serem verificados.

A proposta do *benchmark* propriamente, procura testar os SGBDs com operações do tipo bancárias. Da mesma forma como ocorre com o *benchmark* de Débito e Crédito, ele possui um requisito quanto à qualidade das transações: o tempo de resposta para 90% das transações deve estar abaixo de 2 segundos.

3.5 Resultados de *benchmarks* conhecidos

Nos diversos relatórios e artigos reunidos durante a elaboração deste trabalho são apresentados resultados de testes usando as metodologias em questão. Para as pessoas mais interessadas, indicamos como referência quais testes estão reunidos nestes trabalhos: [BITT83] analisa 4 sistemas (INGRES, DIRECT, IDM500, ORACLE) utilizando o *benchmark* de Wisconsin . No relatório técnico da Tandem Corporation [TAND88] se avalia o desempenho do sistema "NonStop SQL". Comparações entre ORACLE e INGRES podem ser encontradas em [COLL87]. O sistema IDM500 também foi testado em [HAWT85], usando algumas variações no *benchmark* de Wisconsin. Stonebraker [STON82] analisa o desempenho do sistema INGRES distribuído, em rede local e faz uma comparação com a versão convencional (centralizada). Em Duhl [DUHL88], quatro sistemas são comparados (INGRES, Unify, RAD-Unify, VBase Relacional e VBase Object) usando o *benchmark* SUN. Resultados mais recentes, usando o *benchmark* de Wisconsin podem ser obtidos em [BITT89], para ambientes IBM4341, IBM4381 e Teradata ("Teradata Database Machine"). Em [ANDE89], utiliza-se o *benchmark* da Tektronix para se avaliar dois sistemas de bancos de dados orientados a objetos: *VBase* (Ontologic) e *GemStone* (Servio Logic).

```
Read Aid, Tid, Bid, Delta from terminal
BEGIN TRANSACTION
    Update Account where A_ID = Aid:
        Read A_Balance from Account
        Set A_Balance = A_Balance + Delta
        Write A_Balance to Account
    Write to History:
        Aid, Tid, Bid, Delta, Time_stamp
    Update Teller where T_ID = Tid:
        Set T_Balance = T_Balance + Delta
        Write T_Balance to Teller
    Update Branch where B_ID = Bid:
        Set T_Balance = T_Balance + Delta
        Write T_Balance to Branch
COMMIT TRANSACTION
Write Aid, Tid, Bid, Delta, A_Balance to terminal
```

Aid (Account_id), T_id (Teller_id), *and* Bid (Branch_id) *são chaves das tabelas correspondentes. O valor* Delta *é movimentado na conta (*Account*). O valor de* Branch_id, *na mensagem de entrada é o identificador da agência na qual está localizado o caixa automático (*Teller*).*

Figura 3.1: Perfil da transação TP1

Capítulo 4

A Metodologia de Benchmark Especializado

Neste capítulo apresentamos uma metodologia de análise de desempenho de banco de dados baseada em *benchmark especializado* ou, mais resumidamente, *Metodologia de Benchmark Especializado* (MBE).

4.1 Introdução

A MBE utiliza um *modelo de representação* do sistema de banco de dados que pode ser parametrizado pelo usuário, projetista de aplicações ou gerente de um SGBD, através de informações de uma aplicação específica, ou de um conjunto típico de aplicações, que ele encontra em seu ambiente. O modelo de representação é estruturado em *níveis de abstração* que vão desde os níveis que procuram representar as estruturas mais abstratas de um sistema de bancos de dados, até os níveis que representam as características mais físicas.

Ela pode ser aplicada em diversas situações onde se necessita analisar o desempenho de um SGBD, tais como: seleção de diferentes alternativas de projeto de banco de dados (ex. escolhas de índices), processo de escolha de um SGBD ou ainda na determinação da configuração de parâmetros físicos do SGBD (ex. tamanhos de bloco, gerência de *buffer-pool* etc.).

Um dos pontos fortes da MBE é sua portabilidade, que deve-se à simplicidade da utilização da técnica de *benchmark*; que pode se aplicada em diversos SGBD, sem a necessidade de procedimentos especiais ou de alterações no ambiente que se deseja testar.

As várias metodologias de *benchmark* de bancos de dados encontradas na literatura, [BITT83], [BORA84], [ANON85], [TURB87], [RUBE87], [DEMU85], [TAND88], [ANDE89], em sua grande maioria são orientadas para *benchmark de sistemas* e poucas são orientadas para *benchmark de aplicação*. Mesmo as

metodologias que são orientadas à aplicação, são restritas a apenas um tipo, como, por exemplo, o *benchmark* de Débito e Crédito [ANON85], que é orientado para aplicações bancárias, ou o *benchmark* Sun [RUBE87], que é orientado para aplicações de engenharia.

A MBE é orientada para *benchmark de aplicação*, mas, com a diferença de que é o usuário quem determina qual aplicação será caracterizada. Além disso, ela é uma ferramenta completa para a construção de um *benchmark*, englobando as etapas de concepção, construção e execução, facilitando, dessa forma, a utilização da técnica de *benchmark* na análise de desempenho de sistemas de banco de dados.

Na MBE o usuário parametriza o *benchmark* de acordo com as características de uma aplicação ou de um conjunto típico de aplicações. O *benchmark* obtido pode, posteriormente, ser executado no SGBD por ele escolhido. Trata-se, portanto, de um *benchmark de aplicação*, onde o *script* gerado não é representativo para qualquer tipo de aplicação, mas reflete as características próprias de uma aplicação, ou conjunto de aplicações.

Além do banco de dados e da carga de trabalho, a metodologia também procura representar o *ambiente* de execução do *benchmark*, que pode ser configurado em diferentes níveis de concorrência, propiciando uma representação mais representativa de ambientes de sistemas reais.

As estruturas de representação do banco de dados e da carga de trabalho são suficientemente flexíveis de modo a permitir a representação de um banco de dados razoavelmente complexo, onde pode-se definir tabelas, relacionamentos e *visões* dos usuários.

O conjunto de transações pode ser obtido de uma maneira bastante natural utilizando-se um paradigma de formulários (janelas) como uma estrutura através da qual as transações são caracterizadas. Através desse paradigma pode-se obter transações de razoável complexidade que podem incluir operações de: *joins*, definição de visões, projeções, operações sobre agregados, inclusões, exclusões ou atualizações. As relações obtidas podem ser consultadas utilizando-se qualquer um de seus atributos, sejam eles chaves primárias simples, compostas ou atributos comuns.

A carga de trabalho pode ser caracterizada tanto no ambiente interno ao SGBD quanto no ambiente externo. No ambiente interno cada transação obtida é associada a uma freqüência de execução. O ambiente de execução das transações pode ser modelado em níveis diferentes de concorrência permitindo a avaliação das características do SGBD com relação ao compartilhamento dos dados, gerência dos buffers etc. O objetivo é configurar um ambiente que se aproxime do ambiente utilizado pelo usuário.

O *script* do *benchmark*, obtido ao final, contém o esquema e conteúdo do banco de dados, e o conjunto de transações que representam a aplicação ou conjunto de aplicações do usuário. Este *script* é escrito na linguagem SQL, o que facilita a utilização da metodologia para diferentes SGBDs.

4.2 Objetivos

A *técnica de benchmark de bancos de dados* é uma técnica de análise de desempenho bastante eficiente e relativamente fácil de utilizar [TURB87]. Contudo, muitas vezes a utilização de *benchmarks* genéricos pode não ser suficientemente representativo do ponto de vista do ambiente operacional que um determinado usuário possua.

O principal objetivo da MBE é tornar-se uma ferramenta de suporte para análise de desempenho de sistemas de bancos de dados, podendo modelar convenientemente o ambiente operacional do usuário.

A ferramenta pode ser utilizada pelo usuário que desenvolve aplicações e deseja avaliar as alternativas que surgem na fase de *projeto de banco de dados*, assim como pelo gerente de sistemas, que deseja avaliar o desempenho do SGBD para um conjunto de escolhas de parâmetros do sistema (*System tunning*), ou ainda para auxiliar o processo da escolha de um SGBD propriamente.

O modelo de representação do banco de dados procura caracterizar um banco de dados com um grau de complexidade bastante elevado, onde podem ser especificadas as tabelas que compõem o banco de dados, relacionamentos, chaves (chaves primárias simples ou compostas) e, também, visões dos usuários.

A metodologia é passível de automatização, como veremos no capítulo seguinte, em que o banco de dados pode ser gerado automaticamente através de programas ou ainda, se o usuário desejar, ele pode fornecer valores para os atributos.

Não é objetivo da MBE, entretanto, oferecer soluções ótimas para problemas de escolha entre alternativas de projeto (ex. escolha de índices) ou auxílio em fases do projeto lógico (ex. normalização), estas tarefas devem ser realizados pelo próprio usuário. Em contrapartida, a metodologia proporciona um ambiente que pode ser facilmente gerado, onde o desempenho do sistema pode ser avaliado convenientemente para cada uma das alternativas sugeridas pelo usuário.

Em outros *benchmarks* [BITT83], [ANON85], [RUBE87], ao contrário, tanto o banco de dados como a carga de trabalho são predeterminadas. Na MBE o usuário tem a liberdade de modificar a estrutura do banco de dados, incluindo novas relações ou modificando as relações já existentes. Além disso, ele tem a possibilidade de modificar os domínios dos dados e a distribuição de valores.

Em um dado momento, os dados podem ser gerados utilizando uma distribuição uniforme e em outro, utilizando uma distribuição não uniforme, tipo Zipf, por exemplo. Assim, pode-se testar a eficiência do otimizador de transações do SGBD sem ter que alterar a estrutura do banco de dados. A carga de trabalho pode ser modelada, especificando-se o nível de concorrência tanto interno ao SGBD quanto externo, em que vários processos podem ser executados concorrentemente.

Um dos principais objetivos da metodologia é obter um *benchmark de aplicação* caracterizado pelo usuário. Ao contrário, em outras metodologias procura-se

obter um *benchmark de sistema* [BITT83], [BORA84], [ANON85], [TURB87], [DEMU85], [TAND88], [ANDE89], ou então procura-se obter um *benchmark* específico para uma determinada aplicação predeterminada [ANON85], [RUBE87].

Em geral, as metodologias mais conhecidas são tipicamente de *benchmarks de sistema*, tal como aqueles que seguem a filosofia de Wisconsin [BITT83], [BORA84], [TURB87] em que o objetivo é obter-se um *benchmark genérico*, válido para qualquer tipo de aplicação. Este objetivo, no entanto, é muito difícil de se conseguir na prática, pois existe uma diversidade muito grande de fatores que caracterizam as aplicações e cada ambiente onde são executadas. Um problema desta natureza já havia sido abordado em Hawthorn [HAWT85], onde se apontam algumas modificações no *benchmark* de Wisconsin [BORA84] para representar com mais realidade o ambiente do sistema IDM-500 (*Intelligent Database Machine*).

Uma outra linha de *benchmarks* se aproxima mais dos *benchmarks* de aplicação. No teste TP1, que faz parte do chamado *benchmark de Débito e Crédito* [ANON85], procura-se representar transações típicas de aplicações bancárias. Já no *benchmark* da SUN [RUBE87], procura-se avaliar sistemas voltados para ambientes de engenharia.

No entanto, no caso do teste TP1 as transações são preestabelecidas e toda metodologia em si, baseia-se em apenas um tipo de aplicação (aplicação bancária). De forma semelhante, o *benchmark* da SUN procura ser representativo para aplicações típicas de ambientes de engenharia onde, geralmente, se utiliza CAD. Ao contrário, na metodologia aqui apresentada, o usuário pode especificar para qual tipo de aplicação se destina o *benchmark*. Além disso, a MBE pretende ser uma metodologia de desenvolvimento de *benchmarks*, esta metodologia pode gerar, entre outros *benchmarks*, o *Benchmark de Winconsin* ou o TP1. Na avaliação da ferramenta que apresentamos no capítulo 7, utilizamos a MBE para gerar o *Benchmark de Wisconsin*.

Em resumo, com a metodologia desenvolvida é possível:

- Avaliar o desempenho de diferentes SGBDs.
- Avaliar diferentes soluções de projeto. Escolhas de índices e *agrupamentos*, normalizações etc.
- Analisar o desempenho do sistema para diferentes distribuições de dados.
- Testar o efeito de diferentes níveis de concorrência no sistema.
- Testar diferentes composições de carga de trabalho com diferentes níveis de concorrência.

4.3 Níveis de abstração

Um sistema de banco de dados pode ser visualizado em vários níveis de abstração. A partir de uma representação mais abstrata do sistema, podemos

transformá-la através dos níveis, obtendo sucessivas representações mais concretas.

Em cada um dos níveis podem existir diferentes alternativas de solução para um dado problema. As decisões tomadas em cada um deles podem, como em muitos casos, afetar o desempenho do sistema como um todo. Pode-se verificar que, nem sempre as melhores alternativas do ponto de vista de uma otimização local, apenas, sejam as melhores alternativas para o sistema. Em muitos casos, a alternativa ótima, que obtém o melhor desempenho em um determinado nível, pode não ser a melhor alternativa do sistema como um todo.

O desempenho de um sistema de banco de dados deve ser obtido considerando-se o funcionamento de um modo global, e não de modo particular para um determinado nível. Tal fato pode ocorrer, por exemplo, quando se faz a escolha de índices para uma relação.

Idealmente, o processamento de qualquer consulta poderia ser mais rápido se existissem índices em todos os atributos de uma relação. Entretanto, esta escolha pode ser muito cara quando for realizado uma atualização, remoção ou uma inclusão, já que os índices também devem ser atualizados. Além disso, o espaço extra ocupado em memória secundária também é muito maior.

Uma das vantagens da representação do sistema através de um modelo estruturado em níveis é a facilidade adquirida de se esconder certos detalhes de representação e se concentrar nas propriedades comuns de um conjunto de objetos que fazem parte de um mesmo nível em particular.

Sempre que possível, a independência entre os níveis é desejável de modo que uma modificação em um nível inferior não interfira nas escolhas definidas nos níveis superiores.

Tal abordagem torna-se útil porque permite o estudo da interação entre as decisões nos diferentes níveis de um modo mais organizado. O número de níveis escolhido depende das facilidades proporcionadas pela ferramenta e sua representatividade. Deve ser claro, no entanto, que na construção de qualquer modelo, deve-se fazer um balanço entre o compromisso com a representatividade do modelo, em termos da quantidade de detalhes que captura, e a crescente complexidade do modelo final obtido.

Existem várias propostas de modelos de representação de sistemas de banco de dados estruturados em níveis [SEVC81], [RAPO86]. Em cada uma delas, para cada nível enfatizam-se diferentes aspectos dos sistemas.

Na maioria das vezes, a separação entre o que deve ser tratado como lógico ou físico ainda permanece em aberto. No entanto, apesar das diferenças de abordagem, a definição de vários níveis de abstração na representação de sistemas de banco de dados é geralmente reconhecida.

Na proposta de Sevcik [SEVC81] a estrutura de um sistema de banco de dados é representada em sete níveis: abstração, representação, implementação, totalização, otimização, configuração e execução. O modelo PROPHET [RAPO86] possui apenas quatro níveis: semântico, esquema, interno ao banco de dados e sistema/hardware. Em ambas propostas, utilizam-se modelos teóricos para a

representação do sistema e teoria de filas para sua execução.

Na MBE escolhemos por fazer uma divisão em três níveis: *nível lógico*, *nível físico* e *nível sistema/hardware*. O quadro da figura 4.1 mostra uma comparação esquemática entre os modelos.

<table>
<tr><td>NÍVEIS</td><td colspan="2">MBE</td><td>PROPHET [Casas86]</td><td>Sevcik [Sevc81]</td></tr>
<tr><td rowspan="2">LÓGICO</td><td rowspan="2">modelo lógico de dados</td><td rowspan="2">transações (janelas)</td><td>semântico</td><td>abstrato</td></tr>
<tr><td>esquema</td><td>B.D. lógico</td></tr>
<tr><td rowspan="4">FÍSICO</td><td rowspan="4">modelo físico B.D.</td><td>Ambiente</td><td rowspan="4">interno</td><td>B.D. físico</td></tr>
<tr><td>externo</td><td rowspan="2">"data units"</td></tr>
<tr><td>interno</td></tr>
<tr><td>aplicação</td><td rowspan="2">E/S físico</td></tr>
<tr><td rowspan="4">sistema "hardware"</td><td colspan="2">benchmark</td><td rowspan="4">sistema "hardware" (teor. filas)</td></tr>
<tr><td colspan="2">processos concorrentes</td><td rowspan="3">"device loadings" (teor. filas)</td></tr>
<tr><td colspan="2">SGBD</td></tr>
<tr><td colspan="2">sistema operacional</td></tr>
</table>

Figura 4.1: Comparação entre modelos de representação.

No nível lógico da MBE são especificadas estruturas mais abstratas que não são representadas diretamente sobre o SGBD, mas que podem ser transformadas e refinadas em informações mais concretas no nível físico. Fazem parte deste nível, por exemplo, a definição do esquema do banco de dados e a descrição das transações.

No nível físico estamos interessados na definição de parâmetros que nos ajudem a construir os dados, tais como tipos de dados, formatos e distribuições de valores. A formação dos índices e dos agrupamentos também são realizadas neste nível.

Com essa divisão entre lógico e físico, podemos alterar as informações, digamos, no nível físico sem ter que modificar as informações obtidas no nível lógico. Assim pode-se avaliar diferentes alternativas para a solução de proble-

mas no sistema.

No nível de sistema/hardware representamos tanto a carga de trabalho interna ao SGBD quanto externa. Os resultados do modelo são obtidos por técnicas experimentais utilizando *benchmark* de banco de dados.

Na MBE não se procura favorecer nenhum modelo de dados em particular. O modelo relacional foi escolhido apenas com o interesse de representar os dados de uma maneira padronizada, de modo a facilitar o seu entendimento. A MBE também pode ser utilizada em qualquer outro modelo de dados sem perda de generalização.

Além disso, a MBE não procura resolver nenhum problema de escolha entre diferentes alternativas de projeto. O que se oferece, em contrapartida, é um modelo que permite associar escolhas de projeto com valores de medidas de desempenho.

O modelo é passível de automatização, como veremos no capítulo adiante, através da contrução de uma ferramenta de apoio ao usuário projetista de banco de dados, permitindo a obtenção de um ambiente facilmente configurável e flexível, onde podem ser testadas soluções de implementações assim como realizar comparações entre o desempenho de diferentes SGBDs, para uma mesma aplicação prefixada.

4.4 O sistema de banco de dados na MBE

O objetivo do modelo de representação do sistema de banco de dados, utilizado na MBE, é fornecer uma estrutura que pode ser caracterizada pelo usuário de acordo com as informações que ele obtém da sua aplicação e do ambiente de *software* e *hardware* em que será executada.

Basicamente, o *Modelo de Representação do Sistema* divide-se em um *Modelo de Representação dos Dados* e um *Modelo de Carga de Trabalho*. Estes modelos são estruturados hierárquicamente, em diferentes níveis de abstração.

A divisão em níveis, do modelo de representação do sistema, é desejável, pois, a independência entre os níveis, assim proporcionada, facilita a avaliação de alternativas dentro de cada nível em particular e sua influência sobre o desempenho do sistema como um todo. Por exemplo, podemos avaliar o desempenho do sistema para diferentes distribuições dos dados sem ter que alterar a caracterização das transações que compõem a carga de trabalho.

O *modelo de representação dos dados* é estruturado em dois níveis de abstração, um lógico e outro físico. O *modelo lógico* se preocupa com a representação em alto nível do esquema do banco de dados e o *modelo físico* se preocupa com uma representação concreta do modelo lógico em termos de tipos de dados, valores do domínios, formatos dos dados, distribuições, índices etc.

O *modelo da carga de trabalho* é divido em um *modelo de descrição* de transações e um *modelo de execução*, que trabalham de forma integrada para fornecer o conteúdo das transações e o ambiente de execução do *benchmark*.

O *modelo de descrição* de transações procura obter uma caracterização do conteúdo e formato das transações. A um nível mais geral, podem ser representadas transações de *consulta* ou *inclusão*. As transações de consulta podem ser: *consultas simples, consultas com atualização*, ou, ainda, *consultas com remoção*.

No *modelo de execução* podem ser representados diferentes níveis de concorrência, aos quais, o sistema pode estar submetido. Estes níveis podem ser relacionados ao *ambiente externo* do SGBD, *ambiente interno, aplicação* e *transação*. No ambiente externo pode ser especificado o nível de concorrência a nível do SGBD ao passo que, no ambiente interno, pode ser configurado o nível de concorrência a nível das aplicações, através do número de aplicações que são executadas simultaneamente no SGBD. O nível de aplicação procura especificar cada uma das aplicações do usuário, o número de usuários simultâneaos, quantidade de transações etc. Finalmente, o nível de transação procura caracterizar o perfil da utilização das transações provenientes das aplicações dos usuários, fornecendo as freqüência de utilização de cada tipo de transação.

A descrição do sistema é composta por vários objetos de modelagem: relações, atributos, domínio de valores, aplicações, transações etc, que são organizados e armazenados em um banco de dados em memória secundária, constituindo-se no *dicionário de dados* do sistema. À medida em que o sistema necessita de um objeto de modelagem, os dados correspondentes a este objeto são recuperados do dicionário de dados.

4.5 O banco de dados

Para a representação do banco de dados usamos um *modelo de representação dos dados* estruturado em dois níveis: *nível lógico* e *nível físico*, que passamos a descrever.

4.5.1 Modelo lógico do banco de dados

No *nível lógico*, estamos interessados em obter, a partir do mundo real, apenas a parte que nos interessa representar no sistema a ser construído (aplicação do usuário). Este processo se faz através da definição de características estáticas das aplicações que basicamente se preocupam em descrever os dados e o esquema do banco de dados.

Para auxiliar a obtenção das *características estáticas* do banco de dados escolhemos o *Modelo de Entidades e Relacionamentos* (MER) [CHEN77]. O MER se baseia na percepção do mundo real como sendo constituido por um grupo básico de objetos e por relacionamentos entre estes objetos. Sua utilização facilita o projeto de bancos de dados, permitindo a especificação de um esquema de empreendimento. Tal esquema representa a estrutura lógica global do banco de dados desejado.

Basicamente, uma modelagem utilizando o MER permite a identificação dos

objetos, ou *entidades* como são chamadas no MER, e os relacionamentos entre estas entidades. Uma entidade é representada por um conjunto de *atributos*, onde cada atributo possui um *domínio* de valores que são permitidos. Relacionamentos podem existir entre uma ou mais entidades e também podem possuir atributos.

Sobre o MER podem ser definidas certas *restrições de integridade* que devem ser obedecidas na obtenção (geração) dos valores para as relações no banco de dados.

Uma restrição importante é a *cardinalidade de mapeamento*, que expressa o número de entidades ao qual outra entidade pode estar associada via um relacionamento. A cardinalidade de mapeamento pode ser: *um-para-um*, *muitos-para-um*, *um-para-muitos* e *muitos-para-muitos*.

Outra classe de restrições importantes são as *dependências existenciais*. Especificamente, elas querem dizer que, se, por exemplo, x é existencialmente dependente de y, então isso significa que, quando y é removido, então x também deverá ser removido.

Após obtermos esta representação dos dados a um nível mais *semântico*, estas informações devem ser mapeadas em uma representação esquemática no SGBD existente. Para facilitar a compreensão da metodologia, escolhemos o *modelo de dados relacional* como meio de representação dos dados. Não obstante, a metodologia aqui apresentada pode ser aplicada em outros SGBDs que não se baseiam no modelo relacional.

Uma das tarefa importante nesta fase é a identificação de *chaves primárias*. Conceitualmente, entidades individuais e relacionamentos são distintos, mas, na concepção do banco de dados, a diferença entre as ocorrências de uma mesma entidade deve ser expressa em termos de seus atributos.

Uma chave primária identifica uma ocorrência em uma entidade ou em um relacionamento entre entidades unicamente. As chaves primárias podem ser formadas por apenas um atributo, *chave simples*, ou podem ser formadas por mais de uma atributo, *chaves compostas*.

Outro conceito igualmente importante é o de *chave estrangeira*. Em geral, uma chave estrangeira é formada por um atributo (ou composição de atributos) de uma entidade cujos valores devem pertencer ao mesmo domínio dos valores de uma chave primária de uma outra entidade a qual a entidade está relacionada.

Seguindo, para cada atributo de cada entidade deve ser definido o seu domínio de valores. Em princípio, cada atributo de cada relação possui um domínio de valores. No entanto, alguns atributos podem possuir domínios distintos ou então podem compartilhar o mesmo domínio. A possibilidade de compartilhamento do mesmo domínio é desejável pois permite que possam ser realizadas operações mais complexas, tal como *joins*, por exemplo. No caso de uma operação deste tipo, para que ela tenha coerência, os atributos envolvidos na cláusula de junção devem possuir o mesmo domínio.

A metodologia deve garantir que o esquema de entidades e relacionamentos obtidos seja convertido para a linguagem de esquemas disponível no SGBD que

está sendo avaliado. Neste processo de conversão, as entidades são representadas por relações e os atributos das entidades são representados por atributos das relações.

Os relacionamentos, de acordo com a sua cardinalidade, poderão ser traduzidos em novas relações, utilizando chaves estrangeira correspondentes às chaves primárias das relações definidas pelo relacionamento, para o caso do relacionamento N:M. Ou para relacionamentos N:1 e 1:N, onde se utiliza uma chave estrangeira incluida em alguma das relações que participam do relacionamento.

Além disso, o usuário pode definir *visões* para melhor representar a sua modelagem de dados, simplificando a complexidade da estrutura do banco de dados.

A linguagem escolhida para descrever o esquema do banco de dados é a SQL. Esta escolha se deve ao fato da SQL ser uma linguagem padronizada e que se encontra disponível em muitos SGBDs atuais, facilitando a adaptação da metodologia para cada SGBD.

O *benchmark* a ser obtido através da ferramenta que implementa a metodologia, como mostraremos no capítulo seguinte, baseia-se no *modelo relacional*. As definições de esquema, geração das tabelas e representação das transações no *benchmark* são escritas em SQL, e caso o usuário esteja utilizando um banco de dados que não disponha de SQL, deve-se realizar, então, uma tradução para a linguagem disponível no sistema que se deseja testar.

Essa etapa de tradução do modelo lógico para um esquema, em termos do SGBD escolhido, é bastante complexa. Muitas metodologias de projeto de bancos de dados, algumas incluindo até técnicas e ferramentas automatizadas, já foram propostas [NOGU89], [MCLA88]. O objeto destes trabalhos, em muitos casos, é fornecer uma solução ótima para o problema de representação do modelo lógico para o esquema disponível no SGBD escolhido, fazendo-se a normalização automática, criando-se relações particionadas ou agrupando-se, para que se obtenha um melhor desempenho.

Contudo, apesar da importância desta fase de projeto e da qualidade dos trabalhos encontrados na literatura, muitos problemas ainda não têm fácil solução e muitas das decisões ainda devem ser sugeridas pelo projetista, considerando circunstâncias especiais que podem ocorrer. A MBE, por sua vez, não procura resolver este problema, automaticamente, para o usuário. Ele próprio (o usuário) deverá fazer a tradução manualmente ou, então, utilizando uma ferramenta adequada.

Resumindo, nesta fase de modelagem lógica do banco de dados podem ser identificados:

- Relações que compõem o banco de dados.
- Relacionamentos entre as relações.
- Atributos das relações.

- Domínios dos atributos e domínios em comum.
- Identificação das chaves primárias.
- Identificação de chaves estrangeiras.
- Identificação de restrições de mapeamento.
- Esquema do banco de dados expresso na linguagem SQL.

4.5.2 Regras de integridade do banco de dados

Os valores gerados sintéticamente devem estar de acordo com a estrutura do esquema obtido. A cardinalidade dos relacionamentos, a unicidade de chaves primárias e compostas, a integridade existêncial de chaves estrangeiras, devem ser mantidas nos valores obtidos. Para isso, foram definidas algumas *regras de integridade*.

Na MBE estas regras de integridade são as seguintes:

1. *Regra de Unicidade*: se um atributo possui valores únicos na relação então:

$$N \geq Card(R) \tag{4.1}$$

onde N é o número de valores distintos do atributo e $Card(R)$ é a cardinalidade da relação.

2. *Regra da Chave Primária*: para que exista chave primária, então:

$$MAXKEYS \geq Card(R) \tag{4.2}$$

onde: $MAXKEYS$ é o número máximo de valores distintos da chave primária.

3. *Regra de Zipf*: para que seja possível uma distribuição do tipo Zipf, então, devemos verificar:

$$Card(R) \geq N_a \times H_n \tag{4.3}$$

onde: N é o número de valores distintos do atributo e H_n é calculado pela equação 2.2. Para se garantir que o elemento de menor freqüência do domínio do atributo apareça ao menos uma vez.

4. *Regra da Chave Estrangeira*: no caso de um atributo ser chave estrangeira, então, devemos verificar:
   - O atributo origem, referenciado, deve ser chave primária, ou seja, deve satisfazer a regra de chave primária.
   - $N = Card(R_{origem})$ i.e. o número de valores distintos (N) deve ser igual à cardinalidade da relação origem.

4.5.3 Modelo físico do banco de dados

No *nível físico* as informações obtidas no nível lógico são utilizadas para construir o banco de dados. Isso pode ser realizado obtendo-se valores para as relações (seus atributos), contidas no esquema obtido anteriormente, através da caracterização mais detalhada dos dados, em termos dos domínios relacionados e de descrições suplementares dos atributos. Além das informações sobre os dados, devem também ser informadas as estruturas de índices e dos agrupamentos.

A associação dos atributos com seus respectivos domínios já foi realizada no passo anterior. Nesta etapa, estaremos interessados na descrição completa dos domínios dos atributos.

Inicialmente, para cada domínio deve-se determinar o seu tipo de dado ao contrário de outras metodologias, onde o usuário não possui a liberdade de escolha do tipo de dado para o atributo (tipo de dados fixos).

Na MBE o usuário pode escolher qualquer tipo de dado: numérico (real ou inteiro) e cadeia de caracteres, além de outros tipos de dados mais sofisticados, que podem possuir funções próprias para a sua manipulação, como é o caso de datas, horários e campos especiais (campos longos).

O objetivo de se permitir a definição de tipos de dados é proporcionar ao usuário uma maior representatividade dos dados gerados, de acordo com a aplicação ou conjunto de aplicações por ele escolhida dentro do SGBD que ele está procurando testar. O usuário pode, por exemplo, testar o efeito de diferentes tipos de dados para se avaliar o desempenho [TURB87].

Além da definição dos tipos de dados, na descrição do domínio devem ser incluídas as definições de formatos (tamanho fixo ou variável), valores mínimos e máximos, limites de tamanho e número de valores distintos.

A capacidade de definir formatos de tamanho variável é bastante interessante porque permite inserir no modelo de análise de desempenho uma variável que, usualmente, é mantida constante em modelo teóricos e modelos de simulação, o *tamanho de página*, e assim contrariar propositalmente certas hipóteses simplificadoras que levam em conta um comportamento uniforme dos dados, verificando sua influência sobre o desempenho do sistema [CHRI84].

Na metodologia de Wisconsin [BITT83], ao contrário, o tamanho dos atributos é sempre constante, como é o caso de atributos do tipo *string*, que possuem um tamanho fixo de 52 *bytes*. Além de tratar-se de um atributo de tamanho fixo, especialistas da área admitem que este tamanho é muito longo para aplicações usuais, onde os tamanhos de *strings* mais utilizados possuem formato fixo ou variável, com tamanho entre 20 a 30 caracteres [BITT87].

A definição do número de valores distintos dentro do domínio é necessária para permitir a definição de distribuições de dados para o atributo. As distribuições permitidas na MBE podem ser uniformes ou não uniformes (assimétricas).

No *benchmark* de Wisconsin, por exemplo, só se empregam distribuições uniformes (ex. para os atributos: hundred, thousand, string4 etc.). No entanto,

as distribuições de dados em bancos de dados reais são geralmente assimétricas. Atributos tais como: nomes de pessoas, nomes de localidades, de ruas etc. geralmente possuem valores com distribuição não uniforme.

Assumindo apenas hipóteses teóricas de uniformidade, podemos obter uma estimativa incorreta da seletividade e estimar um desempenho que está muito longe da realidade.

A vantagem em modelar distribuições não uniformes é permitir a verificação da capacidade do otimizador de transações em fazer estimativas mais realistas para as escolhas dos planos de acesso, que levem em consideração a assimetria dos valores dos dados.

No *benchmark* de Turbyfill [TURB87] se utilizam distribuições de dados assimétricas (distribuições do tipo Zipf) para se testar otimizadores de transações. Na MBE também é possível caracterizar distribuições não uniformes, do tipo Zipf, por exemplo. Além disso, permite-se também que o próprio usuário possa descrever as freqüências de ocorrência de cada elemento distinto, se ele possuir estatísticas de seu banco de dados.

Apesar de geralmente os otimizadores de transações não utilizarem hipóteses de distribuições de dados assimétricas, pode-se com isso, mostrar a necessidade e a vantagem ,em termos de desempenho, de se passar a utilizar distribuições desse tipo.

Esse conjunto de informações detalhadas que o usuário pode suprir, por um lado faz com que o *benchmark* gerado seja mais representativo, mas por outro lado, um dos objetivos da metodologia é facilitar a configuração do *benchmark* pelo usuário.

Existem situações em que o usuário pode não dispor de tantos detalhes, tais como: distribuições de dados, formatos etc. ou então, quando ele está interessado apenas em ter uma noção aproximada do desempenho. Nesses casos a metodologia prevê a atribuição de valores *default*, que poderão ser assumidos se não forem informados.

As condições para a associação de valores *default* podem ser as mais variadas, tais como: a distribuição de valores do tipo Zipf para atributos do tipo *string*, valores uniformes para atributos numéricos, quantidades de atributos em cada relação etc. Podemos, ainda, considerar as descrições *default* como sendo as descrições do *benchmark* de Wisconsin, ou seja, na ausência de maiores detalhes, o *benchmark* de Wisconsin pode ser assumido como padrão.

Após a caracterização do domínio de valores passamos a caracterizar por completo cada atributo das relações. Em nosso modelo, as descrições de atributos e de domínios devem ser separadas de modo que mais de um atributo possa utilizar o mesmo domínio. Para a especificação dos atributos devem ser informados: a distribuição dos dados, a disposição dos dados sobre as relações, a ordem de colocação no *agrupamento* (se ele participa de algum) e se o atributo aceita valores nulos.

Além de especificar a distribuição dos dados sobre os valores dos atributos, o usuário também pode determinar a disposição dos valores sobre as tuplas

(agrupamento). Na MBE, o usuário pode escolher entre uma disposição de dados aleatória ou seqüencial, e analisar a influência deste fator sobre o desempenho do sistema. No caso do *benchmark* de Wisconsin, existem atributos com disposição aleatória (ex. unique1, stringu1 etc.) e atributos com disposição seqüencial (ex. unique2 e stringu2). Entretanto estes fatores são predeterminados para cada atributo e, ao contrário, na MBE é o usuário quem escolhe estes fatores.

A ordem de colocação dos atributos nos agrupamentos pode influenciar o desempenho do SGBD. Na MBE, não se sugere nenhuma solução com relação à determinação da formação de agrupamentos, automaticamente. Trata-se de um problema bastante semelhante ao da escolha de índices e será solucionado com uma abordagem semenhante, como mostraremos adiante.

Para completar, o usuário deve escolher os índices e a ordem de formação de índices, no caso de existirem índices compostos. Este problema, no entanto, não é nada trivial. Uma boa escolha de índices pode acarretar um melhor desempenho do SGBD quando no processamento de uma transação, entretanto, também pode existir um número muito grande de possíveis soluções para uma dada configuração de carga de trabalho e banco de dados [MELO89], [FINK88]. Se forem realizadas determinadas escolhas que não estejam sintonizadas com o otimizador de transações, estas escolhas podem simplemente não serem aproveitadas durante o processamento das transações, desperdiçando-se espaço em memória secundária e, além disso, encarecendo a execução das operações de inclusão ou atualização no banco de dados.

Na MBE, o problema da escolha dos índices - i.e. a determinação dos atributos que devem possuir índices e quais participam de um mesmo índice (índice composto) e em qual ordem - deve ser resolvido pelo próprio usuário. Para este tipo de problema já foram desenvolvidas metodologia específicas para esta finalidade [MELO88], [FINK88], que poderão vir a ser utilizadas como complementação à metodologia aqui apresentada.

4.6 A carga de trabalho

A determinação da carga de trabalho é muito importante para o sucesso de qualquer metodologia de análise de desempenho. Sua compreensão é extremamente útil para o projeto de sistemas, trazendo implicações tanto na avaliação do *hardware* como na avaliação do *software*. Um estudo mais apurado da carga de trabalho em sistemas reais pode permitir que os SGBDs sejam testados mais eficientemente e melhor sintonizados para o desempenho.

O verdadeiro mérito de várias propostas de *hardware* ou *software* deve ser quantificado com cargas de trabalho realistas, que incluam transações simples e também transações mais complexas envolvendo operações de *joins* com vários tipos de predicados em diferentes níveis de complexidade e operações sobre agregados [YU89]. Pois, se as avaliações somente contiverem características de uniformidade, as avaliações resultantes podem não ser representativas.

Quando os otimizadores de transações procuram resolver o problema da determinação de caminhos de acesso durante a execução das transações, por exemplo, devido ao motivo de desempenho, eles, geralmente, fazem suposições de comportamento uniforme dos valores dos atributos [SELI79]. Estas suposições de uniformidade, no entanto, podem levar à determinação de planos de acesso não otimizados quando na presença de assimetria nos dados e causar um desempenho abaixo do esperado.

Idealmente, a carga de trabalho mais representativa para um dado sistema seria obtida através de um monitoramento do ambiente em produção [MAGA81], [YU89]. Tal abordagem, contudo, é por demais trabalhosa, ou até mesmo ela pode ser impossível de se realizar, devido às condições impostas pelo ambiente: não ser permitido que o sistema seja interrompido, ou que desempenho normal não possa ser afetado, ou, ainda, podem existir restrições quanto à segurança dos dados [GOTL80].

De modo a simplificar este processo ou então pelo simples fato do sistema a ser testado ainda não ter sido construído, como no caso de um projeto de banco de dados, normalmente se utilizam hipóteses simplificadoras para obtenção da carga de trabalho. Estas hipóteses envolvem, geralmente, entre outras considerações, a *uniformidade das transações*, ou seja, as transações referenciam os valores dos atributos uniformemente (com a mesma freqüência) [CHRI84].

No entanto, o comportamento da carga de trabalho é, tipicamente, não uniforme, tal como pode ser verificado em vários trabalhos sobre análise de desempenho. Geralmente, a maioria das transações executadas pelo sistema pertencem a um pequeno conjunto restrito, responsável pela maior parte do consumo de recursos. Em contrapartida, poucas transações que estão fora deste conjunto e raramente são processadas requerem um consumo muito grande de recursos do SGBD [MAGA81], [YU89].

A forma de se obter a carga de trabalho na MBE leva em consideração a não uniformidade da distribuição dos valores dos atributos. Esta característica se verifica quando, na geração de um valor que faz parte de uma cláusula WHERE, o valor gerado deve respeitar a distribuição de dados do atributo correspondente. Ao contrário, na metodologia de Wisconsin [BITT83], todos os valores são uniformemente distribuidos.

Outra caracterísitica que diferencia a MBE da metodologia de Wisconsin se refere à caracterização do ambiente de execução do *benchmark*. Na metodologia de Wisconsin originalmente utiliza-se um ambiente *stand-alone* e monousuário [BITT83]. Posteriormente estas características foram corrigidas em uma versão multi-usuária do mesmo *benchmark* [BORA84]. O *benchmark* AS^3AP [TURB87] segue uma filosofia parecida com a de Wisconsin, mas utilizando um ambiente multi-usuário. A característica do ambiente multi-usuário em um *benchmark* é recomendável porque desta forma pode ser verificada a eficiência das soluções implementadas no SGBD para os problemas de concorrência e compartilhamento dos dados [STON85]. Além disso, um ambiente multi-usuário é mais comumente encontrado nos sistemas de banco de dados.

Na *Metodologia de Benchmark Especializado*, o usuário pode configurar um ambiente de execução, especificando diferentes níveis de concorrência. Podem ser definidos níveis de concorrência ao nível de ambiente externo e do ambiente interno ao SGBD.

No ambiente externo podem ser especificados processos que são executados concorrentemente ao SGBD e que utilizam os mesmos recursos (CPU e E/S).

Internamente, pode ser especificado se várias aplicações são executadas simultaneamente ou se apenas uma aplicação é executada. Para cada aplicação pode ser especificado, ainda, quantos usuários a estão utilizando simultâneamente, se apenas um único usuário ou vários.

Para se obter a carga de trabalho, na *Metodologia de Benchmark Especializado* utilizam-se dois modelos que representam respectivamente um *modelo de descrição* e um *modelo de execução* de transações.

O *modelo de descrição* de transações se preocupa com a obtenção do conteúdo e tipo das transações. Ele utiliza um paradigma de utilização do banco de dados que procura facilitar a caracterização por parte do usuário, já que a definição de formulários, ou de janelas, é um conceito largamente utilizado e bastante natural.

O *modelo de execução* é estruturado em quatro níveis: *ambiente externo*, *ambiente interno*, *aplicação* e *transação*. Nos três primeiros níveis se especifica o nível de concorrência em termos do número de processos sendo executados concorrentemente. No quarto nível se especifica o perfil de utilização das transações através de suas freqüências de execução.

Esta separação em diferentes níveis permite a inclusão de características comuns a um nível em particular, onde não se necessita modificar, por exemplo, o conteúdo das transações, para se obter uma nova carga de trabalho com uma configuração diferente do nível de concorrência existente no sistema. Esta facilidade pode ser importante quando se deseja, por exemplo, testar os módulos do SGBD que são responsáveis pelo compartilhamento de recursos e gerência de *buffers*.

4.6.1 Modelo de descrição de transações

No *modelo de descrição* de transações podem ser representadas transações de dois tipos: *consultas* e *inclusões*. As consultas pode ser *consultas simples*, *consultas com atualização* ou *consultas com remoção*.

Para facilitar a obtenção do conteúdo e da forma das transações do usuário, utiliza-se um *paradigma* de utilização de bancos de dados através de formulários (ou janelas), que apresentamos com mais detalhes na próxima seção. Este esquema fornece uma estrutura que pode ser facilmente compreendida pelo usuário e através da qual se obtém uma caracterização bastante realista das transações do usuário.

Este modelo de descrição, utilizando o paradigma de formulários, pode ser encarado como uma descrição lógica das transações, que procura representar as

características dinâmicas do banco de dados, ou seja, procura descrever, basicamente, como se utilizam e manipulam os dados no modelo de representação dos dados, definido anteriormente.

Paradigma de formulários

O paradigma de utilização de banco de dados através de formulários vem sendo, atualmente, largamente utilizado como modelo de projeto de interfaces de SGBDs. O conceito e a utilização de formulários, ou janelas, são amplamente recompensados pelas facilidades na definição e construção de uma aplicação para o usuário.

A linguagem QBE [1] utiliza um conceito semelhante, em que o usuário pode realizar uma consulta ao banco de dados, preenchendo com valores como se fosse uma tabela, somente para os atributos que ele deseja pesquisar. O processamento da consulta se encarrega de realizar a seleção apropriada, devolvendo os atributos restantes da tabela, não preenchidos pelo usuário, com os valores obtidos do banco de dados.

Outros exemplos de utilização deste paradigma são a ferramenta Fillin [WART86] e SQL*Forms [BENN86]. O paradigma de formulários é na realidade um conceito muito utilizado atualmente.

Antes de tudo, um *formulário* é um objeto de fácil compreensão para o usuário. Ele pode ser facilmente definido, pois o conceito de formulário é largamente utilizado na vida prática, e se ajusta muito bem na representação de relações (tabelas) e suas tuplas (linhas das tabelas).

Basicamente um formulário pode ser composto por:

- *Janelas*: uma janela está geralmente associada a uma visão (visão base) à qual são realizadas a maior parte das transações. Durante a operação sobre uma janela, muitas vezes, é necessário, também, consultar outras visões a ela relacionadas. Uma janela pode exibir um ou mais registros da visão base, a cada vez.

- *Campos*: uma janela é composta de campos que exibem os valores de uma visão. Em geral, correspondem a valores de atributos da visão base. Os campos podem também ser campos temporários, que são utilizados para algum armazenamento de algum cálculo intermediário para facilitar o entendimento do formulário, ou podem ser campos que exibem dados de outras visões que não a visão base.

- *Valores*: são os itens de informação propriamente.

Entre as vantagens do uso de um formulários eletrônico podemos citar:

[1] *Query by Example.*

- *Facilidade de utilização*: os dados podem ser digitados diretamente nos campos corretos e inseridos no banco de dados; pode-se visualizar, atualizar, remover um ou vários registros no mesmo formulário de uma só vez. A condição de uma consulta pode ser colocada diretamente nos campos onde ela é desejada.
- *Facilidade de concepção*: a linguagem para definição de um formulário é facilmente assimilável, inclusive por pessoas não especializadas em programação. Além disso, é facilmente integrado com relações do banco de dados.

Além dessas facilidades, um sistema baseado em formulários pode ainda:

- Facilitar a validação de dados.
- Restringir inserções ou atualizações a determinados campos.
- Manutenção da integridade entre as tabelas do banco de dados.

Na concepção de formulários, as transações são desenvolvidas centradas em uma tabela (visão) base sobre a qual são realizadas interações, ou seja, para cada tupla (registro) da visão base que sofre alguma manipulação, vários outros registros de tabelas (visões) relacionadas à visão base são consultados, criados, atualizados ou removidos. Este esquema procura definir um modelo genérico que pode ser parametrizado pelo usuário.

Para cada janela desta estrutura, existe uma visão associada (ou tabela) sobre a qual são realizadas as várias operações (consulta, inserção, atualização ou remoção). Podemos mostrar na janela uma ou várias linhas da visão associada, onde, em cada campo da janela, é mostrado o valor de um atributo. Desta forma, podemos imaginar a aplicação como sendo formada por várias janelas interligadas, onde cada janela pode visualizar uma parte de uma tabela.

Em muitas situações pode ser interessante modelar uma transação utilizando mais de uma janela. Enquanto na *janela principal* pode ser mostrado, por exemplo, um registro de um pedido de compra, tal como na figura 4.2, com informações sobre número do pedido, data de compra, número do cliente etc., a segunda janela pode, por exemplo, mostrar detalhes dos itens do pedido, que se encontram em outra tabela.

Este paradigma, além de permitir a configuração de transações bastante complexas, utilizando visões, também facilita, a nível de interface, a obtenção de características relacionadas ao banco de dados do usuário. Por exemplo, para uma dada janela, o usuário pode especificar através de qual campo da janela ele irá buscar dados na base de dados. Assim, ele pode especificar diferentes tipos de transação para uma mesma janela. Associado a informação para cada janela, também pode ser obtido o número médio de registros recuperados para aquele tipo de consulta. Através dessa informação, o banco de dados pode ser construído para refletir a *seletividade* [2] fornecida, ou seja, o modelo de

[2]Número de registros qualificados em uma consulta

descrição de transações suplementa as infromações que porventura não foram obtidas no modelo de dados. Se a seletividade for de apenas um registro, pode-se deduzir que o atributo referenciado possui apenas valores únicos, o qual pode ser escolhido para ser a chave da relação.

Convém neste ponto que façamos uma observação: uma transação, na realidade, consiste em um conjunto de operações que devem ser tratadas como uma unidade. Sem perda de generalidade, utilizamos até este ponto na MBE, o conceito de transação como se fosse o mesmo conceito de um comando (comando SQL) simplesmente; pois, assim, as transações poderiam ser representadas como comandos de consulta ou comandos de inclusão. Os comandos de consulta, por sua vez, poderiam ser: consultas simples, consultas com atualização e consulta com remoção.

Desta forma, assumimos que cabe ao usuário a definição de suas transações de modo que a integridade do banco de dados seja preservado. Quer dizer, no caso de uma remoção de uma tupla em que existem restrições quanto à integridade existencial, as tuplas a ela relacionadas também deverão ser removidas, de modo a manter a integridade do banco de dados. Esta tarefa, portanto, cabe ao usuário, pois é ele quem deve controlar a integridade, da mesma forma como ocorre na maioria dos SGBDs atualmente.

4.6.2 Modelo de execução da carga de trabalho

O *modelo de execução da carga de trabalho* é dividido em quatro níveis: *nível externo*, *nível interno*, *nível de aplicação* e *nível de transação*.

Nível de ambiente externo

O *ambiente externo* pode influenciar significativamente o desempenho do SGBD no atendimento às transações dos usuários. Tanto o ambiente de hardware como o ambiente de software influenciam significativamente o desempenho do sistema de banco de dados.

As características de *hardware* onde o SGBD está instalado influenciam o desempenho, tais como: espaço disponível de memória, tempo de acesso à disco, existência ou não de uma arquitetura de *hardware* especial e características de capacidade do *hardware* em geral.

A substituição de determinados componentes por componentes mais eficientes, ou a adição de mais componentes (ex. mais unidades de disco), pode alterar o desempenho do sistema como um todo e conseqüentemente beneficiar o sistema de bancos de dados.

A metodologia aqui apresentada, no entanto, não se preocupa com os fatores externos relacionados à *arquitetura* ou *capacidade do sistema*, apenas se concentra nos fatores relacionados ao ambiente de *software* (interno e externo). O *benchmark* AS^3AP [TURB87] possui este tipo de preocupação com relação

à comparação de sistemas com diferentes capacidades e diferentes arquiteturas de *hardware*.

Em um *benchmark* também é importante avaliar-se o ambiente de *software*, porque geralmente os SGBDs são utilizados em máquinas não dedicadas, onde existem outros processos sendo executados concorrentemente que disputam os mesmos recursos entre si (CPU, memória, disco). Este problema foi estudado, especialmente, no *benchmark* AS^3AP [TURB87].

Na *Metodologia de Benchmark Especializado*, se inicia a configuração da carga de trabalho, determinando-se o nível de concorrência externa ao SGBD através do número e tipo dos processos concorrentes. O tipo do processo pode ser classificado quanto ao seu perfil de utilização de recursos, tempo de processador e número de operações de entrada e saida (E/S). A partir desse critério, obtém-se uma classificação em quatro tipos de processos, combinando-se, como em [BORA84] os dois fatores citados.

Para se caracterizar cada tipo de processo quanto à utilização de recursos, pode-se escolher, na própria instalação em que os testes serão realizados, um conjunto de programas que se enquadram em cada uma das quatro classes de processos. Assim, pode-se avaliar diferentes tipos de carga de trabalho externa para uma configuração de carga de trabalho interna do SGBD pré-estabelecida.

Nível de ambiente interno

O ambiente interno refere-se apenas à carga de trabalho originária da execução de transações dos usuários do SGBD. Em uma dado momento podem estar sendo executadas diferentes aplicações no SGBD simultâneamente. Em outro momento, apenas um tipo de aplicação pode estar sendo executado. O usuário tem a liberdade de configurar estes parâmetros de forma a melhor representar o seu ambiente. Ele pode representar vários usuários para uma mesma aplicação ou várias aplicações diferentes (ambiente multi-usuário) ou apenas um usuário, utilizando várias aplicações (ambiente mono-usuário). Em algumas situações é interessante fazer testes em um ambiente mono-usuário, pois, assim, pode-se considerar como sendo o melhor tempo de resposta possível para o atendimento daquela transação [BITT83], [BITT87].

Nível de aplicação

Para a configuração da aplicação utilizamos o paradigma de formulários. Este paradigma procura ser bastante representativo para vários tipos de aplicação, como mostraremos mais adiante. Uma aplicação pode ser visualizada como sendo composta por vários formulários, ou janelas, como chamamos na metodologia de *benchmark* especializado. Uma janela pode ser executada várias vezes. Ela também pode passar dados para uma outra janela, que é executada toda vez que a anterior também for. Para cada janela definem-se um ou vários tipos de transações, que são executadas quando a janela é escolhida. A execução de

uma janela significa dizer que são executadas uma ou várias transações dessa janela. Este paradigma é bastante genérico de modo a permitir a representação de vários tipos de aplicações.

Para a aplicação podem ser configuradas várias janelas e à cada janela podem ser associadas freqüências de execução distintas. Além disso, esta estrutura também permite que, dentro de uma aplicação, possa ser caracterizado o nível de concorrência, ou seja, quantos usuários podem utilizar simultaneamente a aplicação no SGBD. A seguir apresentamos com mais detalhes o paradigma de formulários, que é utilizado para auxiliar a caracterização das aplicações.

Nível de transações

A obtenção das transações é extremamente facilitada com a utilização do paradigma de formulários. Para uma dada janela de uma aplicação o usuário, preenchendo apenas um dos campos da janela, determina automaticamente a busca (consulta) na visão associada à janela, pelas informações que não se encontram preenchidas. Isso representa uma consulta do tipo SELECT para um dado valor fixo de um ou mais dos atributos da visão base, cujo valor é calculado automaticamente.

Este mecanismo facilita a definição da aplicação do usuário, onde ele determina, por exemplo, que a freqüência maior de consultas sobre esta janela ocorre para valores inseridos nos campos A, B,ou C com uma freqüência de ocorrência determinada. Ou seja, normalmente digamos 60% das consultas são do tipo 1, onde se utiliza o campo A da janela, e assim por diante. Como o paradigma de formulários pode ajudar a expressar transações bastante complexas, acreditamos que a abordagem utilizada é bastante boa e representativa de qualquer tipo de aplicação tradicional em bancos de dados.

4.6.3 Resumo da carga de trabalho

Resumindo, podemos identificar as seguintes informações que podem ser representdas na configuração da carga de trabalho:

- *Ambiente externo*: determinamos o nível de concorrência externo ao SGBD, através do número de processos que podem ser executados concorrentemente e do tipo dos processos.
- *Ambiente interno*: determinamos o nível de concorrência interna no SGBD e assim verificamos a eficiência das soluções para ambiente multi-usuário no SGBD.
- *Aplicação*: definimos uma aplicação ou conjunto de aplicações com a ajuda do paradigma de formulários. Determinamos o número de janelas que compõe uma aplicação, as visões bases relacionadas e as freqüências de utilização de cada janela. Definimos, também, o nível de concorrência

dentro de uma mesma aplicação, podendo eventualmente utilizá-la em um ambiente mono ou multi-usuário.

- *Transação*: utilizando a paradigma de janelas para facilitar a definição das transações dos usuários, podemos configurar diferentes transações contendo, consultas, inclusões, atualizações e remoções. Para cada tipo de transação pode ser especificada uma freqüência de execução.

Exemplo de utilização de janelas: *A primeira janela mostra 1 registro da tabela base PEDIDO. Os campos mostram ocorrências dos atributos da tabela base. O campo* nome, *mostrando o nome do cliente, pode ter sido obtido em uma tabela relacionada, CLIENTE.*

CONTROLE DE PEDIDO

num_pedido : 2578 — data pedido : 12/01/90

cod_venda : 100 — data entrega : 20/01/90

cod_cliente : 1245-80 — nome : Oficina ABC Com. e Ind.

A segunda janela mostra os registros da tabela base ITENS_DE_PEDIDO, que se relacionam com o registro mostrado na janela anterior. Alguns campos, como sub-total *e* total, *são campos temporários.*

ITENS DE PEDIDO

num. pedido	num. item	descricao produto	preco tabela	qtde	sub-total	qtde estoque
2578	1	carburador	25.000	8	200.000	150
2578	2	filtro oleo	1.200	32	38.400	500
				total :	238.400	

Figura 4.2: Janelas da tabelas: cliente, pedido, itens_de_pedido.

Capítulo 5

Implementação da ferramenta

Este capítulo descreve a implementação de uma ferramenta de análise de desempenho, baseada na MBE apresentada no capítulo anterior. A implementação da ferramenta segue uma estrutura em camadas, sendo que a camada de nível mais baixo (nível de sistema/*hardware*) utiliza a técnica de *benchmark* de bancos de dados.

5.1 Objetivos

A ferramenta de análise de desempenho de banco de dados, obtida a partir da MBE, provê meios ao usuário/projetista de banco de dados para analisar o desempenho de um sistema sob o ponto de vista de uma aplicação ou de um de conjunto típico de aplicações. Não se trata de um analisador genérico de desempenho de SGBDs; no entanto, a ferramtenta permite a obtenção de um ambiente adequado, composto de banco de dados, índices, *agrupamento* e conjunto de transações, onde se pode avaliar e escolher alternativas de projeto de bancos de dados e de parâmetros de configuração do SGBD.

5.2 Características gerais

O *Modelo de Representação do Sistema de Bancos de Dados* utilizado para gerar o *benchmark* é parametrizado pelo usuário através da caracterização de uma determinada aplicação ou um conjunto típico de aplicações, juntamente com as descrições do ambiente computacional onde são executadas. Este modelo é divido em um *Modelo de Representação dos Dados* e um *Modelo de Carga de Trabalho*.

O *Modelo de Representação de Dados* é baseado nas informações supridas por uma diagrama padrão de Modelagem de Entidades e Relacionamentos (MER), onde são obtidos dados qualitativos dos objetos que descrevem o banco de dados. Como o MER é insuficiente para fornecer todas as informações necessárias para configurar um banco de dados com mais detalhes, é requerido que o usuário refine as informações fornecendo mais detalhes (distribuição dos dados, domínio de valores, disposição das tuplas sobre as relações etc.), ou, se não for possível, estas informações são assumidas por *default*.

O *Modelo de Carga de Trabalho* utiliza um paradigma de utilização de bancos de dados baseado em janelas que são parametrizáveis pelo usuário, a partir de informações sobre a aplicação escolhida ou de um conjunto típico de aplicações.

As descrições dos modelos são compostas por vários objetos de modelagem: relações, atributos, domínio de valores, transações, índices etc. que são organizados e armazenados em um banco de dados em memória secundária, constituindo assim um *dicionário de dados centralizado* do sistema. À medida em que a ferramenta necessita um objeto de modelagem, os dados correspondentes a este objeto são recuperados do dicionário.

Inicialmente as informações do Modelo de Representação de Dados e do Modelo de Carga de Trabalho são carregados e armazenados no dicionário de dados da ferramenta. A partir daí prossegue-se a geração do *script* do *benchmark*.

O *script* gerado é composto de uma seqüência de comandos SQL que criam as bases de dados e conjuntos de transações. Por motivos de praticidade, esta seqüência pode ser divida em duas partes; a primeira cria o banco de dados para testes e a segunda representa a carga de trabalho. Esta seqüência de comandos inclui:

- Comandos de criação e definição (DDL) de objetos (relações, visões, índices etc.) do banco de dados.
- Comandos de manipulação de dados (DML): inserção, atualização e remoção de dados.
- Comandos de consulta (select).

Na primeira fase, os comandos da DDL criam os objetos que irão ser representados (relações, índices, *agrupamento* etc.). Em seguida, comandos DML inserem valores para os atributos das relações recém-criadas. Os valores utilizados são obtidos através de algoritmos especiais, que são construídos de acordo com as definições do domínio de cada atributo, contidas no dicionário de dados.

Finalmente, é construída a seqüência de comandos, que constitui a carga de trabalho do sistema propriamente. A carga de trabalho, composta de seleções, atualizações, remoções e inserções, é construida de forma a preservar as propriedades do banco de dados, que se mantem invariantes ao longo da execução das transações. Antes de iniciar a execução da carga de trabalho, pode ser conveniente preservar o conteúdo do banco de dados, salvando-a em um arquivo externo

ao SGBD. Desta forma, ela pode ser utilizada, posteriormente, em outros experimentos, fazendo-se variações na composição da carga de trabalho (misturas de transações).

A implementação da ferramenta foi realizada em Turbo Pascal [PASC87] versão 5.0 utilizando um pacote que implementa funções de banco de dados, *Turbo Pascal Database ToolBox* [TOOL85]. O ambiente de testes utilizado com mais freqüência foi o SGBD *ORACLE* versão 4, executado em um microcomputador tipo PC-AT. Tal ambiente, apesar de não oferecer muitas das vantagens que seriam úteis no desenvolvimento da ferramenta, foi utilizado devido a facilidades de disponibilidade. Além deste ambiente, também foi utilizado, com menor freqüência, o SGBD *ORACLE* versão 5 sob VMS/VAX e a rede de pacotes Renpac (Embratel) como meio de conexão.

5.3 Geração de valores aleatórios únicos

Em várias situações, durante a execução da ferramenta, será necessário utilizar uma fonte segura de números aleatórios únicos, que possa ser utilizada para gerar uma *seqüência de valores aleatórios únicos*. Tais situações podem ocorrer, por exemplo, quando se necessita gerar valores aleatórios sem repetição para um atributo, ou quando se deseja incluir as tuplas nas relações de acordo com uma ordem aleatória.

Podem ocorrer duas formas distintas de utilização desta seqüência de números aleatórios únicos. Na primeira forma, é necessário que os valores sejam provenientes de uma mesma fonte, na segunda, é necessário que os valores obtidos sejam originários de uma nova fonte de valores que ainda não tenha sido utilizada anteriormente. Em ambas as situações, o algoritmo utilizado para a obtenção dos valores é o mesmo.

Uma vez gerada a *seqüência permanente*, ela pode ser armazenada em disco para ser utilizada posteriormente sem a necessidade de uma nova geração. Toda vez que a ferramenta é utilizada, verifica-se inicialmente se a seqüência existe, caso ela já tenha sido gerada, então ela é carregada para memória principal. Quando se necessita de uma fonte de valores aleatórios a cada vez, a seqüência é descartada logo após sua utilização e uma outra seqüência é obtida, sempre que for necessário.

Para a geração da seqüência de valores aleatórios únicos, ou a *coluna de valores aleatórios únicos*, utilizou-se o seguinte algoritmo:

> Um mapa do tipo *bit-map* é construído com o tamanho igual ao número de valores distintos desejado. A seqüência aleatória de valores únicos é obtida pesquisando-se no *bit-map* através de um valor escolhido randomicamente no intervalo de $1..N$; onde N corresponde ao número de valores distintos. Caso o valor escolhido randomicamente já tenha sido escolhido anteriormente, então o *bit-map* deve indicá-lo como ocupado. Caso seja a primeira vez, o valor é incluído

na seqüência e a posição referente no *bit-map* é ocupada. A pesquisa termina, somente, após todos os valores do *bit-map* terem sido escolhidos. Na prática, podemos considerar que, após 90% dos valores terem sido escolhidos, os valores restantes da seqüência podem ser obtidos percorrendo-se o *bit-map* em ordem seqüencial, retirando-se os valores não escolhidos e incluindo-os na coluna de valores aleatórios únicos na mesma ordem em que forem encontrados.

A partir da obtenção de uma *coluna de valores aleatórios*, pode-se obter uma função (*GetRandUnique*), que fornece um valor aleatório único sem possibilidade de repetição, bastando, para isso, percorrer a coluna gerada na ordem seqüencial.

Quando desejamos obter uma distribuição de valores aleatórios únicos para um atributo cujo domínio seja definido em um intervalo entre um valor mínimo e um máximo, percorremos a coluna de valores aleatórios em ordem seqüencial e selecionamos os valores da coluna que estão dentro do intervalo desejado. Assim, a seqüência obtida é gerada de forma determinística, desde que a coluna de valores aleatórios seja sempre a mesma.

A utilização da coluna permanente de valores aleatórios únicos facilita a geração de valores para diferentes atributos que possuem o mesmo domínio. Tornando possível gerar valores para uma chave primária de uma relação e uma chave estrangeira, correspondente, em outra relação.

Para se inserir os valores gerados nos atributos da relação de uma forma aleatória, deve-se obter um valor aleatório único para indicar a posição em que este valor será inserido. Este número aleatório é obtido de uma coluna de valores aleatórios únicos, não permanente, que é gerada para cada atributo, individualmente, que possua disposição aleatória. Por motivos de eficiência, o tamanho da coluna criada pode ser igual a cardinalidade da relação da qual pertence o atributo que está sendo gerado.

5.4 O dicionário de dados

O *Dicionário de Dados* é dividido em duas partes, que representam o *Modelo de Representação dos Dados* e o *Modelo de Carga de Trabalho*, respectivamente (Apêndice B).

5.4.1 Parte 1: Modelo de Representação dos Dados

O *Modelo de Representação dos Dados* engloba todas as descrições do banco de dados utilizado na aplicação do usuário. Fazem parte deste modelo as seguintes relações:

1. *Agrupamento*: relaciona as definições dos *agrupamento* do banco de dados, identificando seus atributos e tipo de dado.

2. *Relação*: armazena as informações dos nomes de relações, uma estimativa da cardinalidade de cada relação e o número de atributos.

3. *Atributo*: descreve cada atributo de uma relação: domínio a que pertence, restrições ao domínio, o número de valores distintos, se aceita valores nulos, qual a distribuição dos valores, a disposição dos valores sobre as tuplas (seqüencial, aleatório), a ordem de colocação no *agrupamento*, valores máximos e mínimos, tamanhos máximos e mínimos. Além destes dados, deve-se informar, também, se os valores são obtidos automaticamente pelo sistema ou se são fornecidos pelo usuário.

4. *Domínio* : cada atributo está associado a um único domínio, mas diferentes atributos podem compartilhar o mesmo domínio. No domínio devem estar as definições de tipos de dados, formatos (tamanho fixo ou variável), valores mínimos e máximos, limites de tamanho e número de valores distintos.

5. *Chave*: os atributos que são chaves primárias devem ser relacionados nesta tabela. Podem estar armazenadas chaves primárias simples e compostas. São identificados subchaves próprias que fazem parte de uma chave composta, e a relação da qual são originárias.

6. *Chave Estrangeira*: relaciona os atributos de uma relação que são chaves estrangeiras, ou seja, não possuem domínios definidos na relação a que pertencem. Os valores destes atributos devem ser chaves primárias de uma outra relação. Desta forma, o domínio de valores é obtido a partir da chave primária original.

7. *Lista*: os valores dos atributos utilizados pelo *benchmark* são construídos pela ferramenta, através de algoritmos apropriados. Além disso, existe também a possibilidade do próprio usuário fornecer este dados através de listas de valores. Cada lista deverá identificar o atributo através do seu nome e do nome da relação à qual pertence. O número de ocorrências de valores para um atributo específico deverá ser igual ao número de valores distintos definidos para o mesmo atributo.

8. *Freqüência*: podem ser utilizadas distribuições uniformes e não uniformes. Para o caso não uniforme, a freqüência de cada valor distinto pode ser obtida automaticamente por algoritmos, utilizando uma distribuição do tipo Zipf. Além disso, o próprio usuário pode fornecer os valores, caso disponha dos valores de freqüência para cada elemento distinto pertencente ao domínio.

9. *Índice*: a utilização apropriada de índices pode incrementar o desempenho do sistema. A ferramenta não auxilia o projeto de índices, porém permite sua configuração para cada relação. Neste caso, o próprio usuário deve identificar as relações, os atributos e a ordem de formação dos índices.

Neste modelo podemos configurar *agrupamentos* de relações, descrever estruturas de índices e identificar para cada atributo o seu domínio de valores e a distribuição dos valores do domínio sobre os atributos. Ainda, podemos definir a disposição das tuplas na relação, de acordo com os valores de alguns de seus atributos, obtendo uma ordenação seqüencial ou aleatória.

Cada atributo possui um domínio de valores que são descritos independentemente dos atributos, permitindo-se que diferentes atributos possam compartilhar o mesmo domínio. Além disso, na descrição dos atributos, pode-se, ainda, fazer algumas restrições sobre o domínio, para se obter, finalmente, o conjunto de valores que irão ocorrer. Estas restrições são quanto ao número de valores distintos do atributo, que pode ser menor que o domínio original, e valores máximos e mínimos. Se temos, por exemplo, um domínio com 1000 valores distintos e um atributo que possua 10% de restrição ao domínio original, então este atributo deverá ter até 900 valores distintos.

5.4.2 Parte 2: Modelo de Carga de Trabalho

O *Modelo de Carga de Trabalho* é formado pelo conjunto de relações:

1. *Aplicação*: relaciona o conjunto de aplicações que fazem parte do *benchmark* e suas freqüências de execução respectivas.

2. *Janela*: uma aplicação pode possuir várias janelas. Para cada janela deve ser especificado o nome da visão associada e mais a condição de set-up expressa pela *cláusula_set-up*, que será explicada mais adiante. Informa-se, ainda, uma estimativa do nº de transações executadas em uma seção na janela (N_s^t), o nº médio de seções utilizadas em uma execução da aplicação (N_a^s) e a freqüência de execuções de transações de consulta ($f_{sel.}$). A freqüência de inclusões é dada por $f_{ins.} = 1 - f_{sel.}$, e o nº de transações de inclusão é obtido por $N_s^t \times f_{ins.}$.

3. *Visão associada*: expressão (em SQL) das visões utilizadas nas janelas.

4. *Campos*: informa os campos da visão associada, nomes de atributos e a relação a que pertencem, para a identificação do domínio.

5. *Seleção*: para uma janela podem existir vários tipos de consulta. Para isso devem ser informados os campos (atributos) que fazem parte de cada *cláusula_tipo*, que faz parte do modelo de consulta genérica, que será mostrado adiante. Cada tipo de consulta possui uma freqüência parcial de execução e freqüências de atualização e remoção.

5.5 Obtenção das relações

A relações são geradas conforme as definições contidas no esquema do banco de dados que se encontram armazenadas no dicionário de dados. Para cada

relação são obtidos os valores de cada atributo individualmente, de acordo com as características definidas em seus domínios.

A partir de uma cardinalidade inicial da relação, para cada atributo é identificado um algoritmo específico que produza toda a coluna de valores com tamanho igual à cardinalidade fornecida.

A geração completa de uma relação é dividida em duas etapas. Na primeira, é definida uma relação temporária, denominada TEMP, que inclui todos os atributos definidos na relação original e mais um atributo denominado NUMSEQ, que possui um valor seqüencial, indicando a posição da tupla dentro da relação. Sobre esta relação geram-se todos os valores dos atributos para a cardinalidade desejada. Posteriormente, na segunda etapa, as tuplas da relação TEMP são copiadas para a relação definitiva.

A primeira coluna na relação TEMP, será a coluna NUMSEQ, que é preenchida com um valor seqüencial de $0..C_r - 1$; onde C_r é a cardinalidade da relação. A finalidade da coluna NUMSEQ é localizar uma tupla dentro da relação, permitindo que os valores gerados por um algoritmo específico, de acordo com as características do atributo, sejam inseridos em uma posição determinada conforme a ordem desejada. Ou seja, é possível o controle da disposição das tuplas sobre a relação de acordo com o valor de um ou mais de seus atributos. No modelo apresentado, são permitidas a ordem seqüencial ou aleatória. Para se inserir um novo valor, basta executar um comando *update* com a condição *where* $NUMSEQ = valor$, onde *valor* é obtido de acordo com a disposição desejada.

Na geração de valores com disposição aleatória, utiliza-se uma coluna de valores aleatórios, que é gerada individulamente para cada atributo. A coluna gerada possui uma cardinalidade igual à cardinalidade da relação que está sendo gerada. Para se obter uma disposição aleatória, basta escolher a posição de insersão do atributo, percorrendo-se seqüencialmente a coluna de valores aleatórios. Para cada atributo de uma mesma relação, uma nova coluna de valores aleatórios deve ser obtida, pois, desta forma, a disposição de um e outro atributo se tornam totalmente aleatórias.

A distribuição dos valores sobre as tuplas de uma relação e a disposição física das tuplas têm uma grande influência sobre o desempenho de um sistema. Sendo assim, é importante que o modelo possa representar estas características mais realisticamente e não assumindo apenas características de uniformidade [CHRI84].

5.5.1 Dados gerados

Os dados gerados pela ferramenta possuem um formato semelhante aos dados utilizados no *benchmark de Wisconsin*. São gerados dados de dois tipos: numérico e cadeia de caracteres (*string*). A diferença entre as metodologias é com relação à utilização. Na MBE, é o usuário quem especifica os parâmetros dos dados, tais como: tamanho do atributo (mínimo, máximo), valores (mínimos e máximos) e formato (fixo ou variável). Ao contrário, no *benchmark de Wis-*

consin os formatos dos dados assim como a estrututra do banco de dados é preestabelecida.

Os dados *numéricos*, como são obtidos em alguns casos através da utilização da *coluna de valores aleatórios*, são limitados pelo tamanho do *bit-map* utilizado, descrito na seção anterior. Além disso, o SGBD também pode impor algumas restrições quanto aos limites dos valores numéricos. Nesta implementação que estamos descrevendo, somente foram utilizados valores do tipo inteiro longo (32 bits). A razão de não se utilizar valores do tipo real deve-se ao fato de ser maior o interesse na distribuição dos valores nos atributos da relação e sua conseqüência sobre o desempenho (operações de E/S), e não a carga computacional exigida em operações de cálculo, como seria o caso de estarmos utilizando valores numéricos de ponto flutuante. O número de operações de E/S possui, sem dúvida, uma maior influência no desempenho de sistemas de arquitetura tradicional do que operações de cálculo.

Os atributos do tipo *string* podem ser sequenciais ou aleatórios, e o formato dos atributos pode ser fixo ou variável. Esta possibilidade de modelar atributos com tamanho variável é bastante interessante, pois contraria hipóteses de uniformidade usualmente empregadas em modelos analíticos [CHRI84].

Os atributos do tipo *string* têm o seguinte formato genérico:

$$\underbrace{X}_{1^{a}\,parte}\ \overbrace{nnn}^{2^{a}\,parte}\ \underbrace{XXX...XX}_{3^{a}\,parte}$$

A 1^{a} parte é composta por 1 caracter fixo "X". A 2^{a} parte é preenchida com um valor numérico e a 3^{a} parte, que pode possuir tamanho fixo ou variável, é formada pela concatenação de caracteres não significativos, com o intuito, apenas, de formar o tamanho final desejado para o atributo.

5.5.2 Geração inicial dos dados

As distribuições de dados para o modelo desenvolvido podem ser uniformes e não-uniformes. Distribuições não-uniformes são representadas por distribuições do tipo *Zipf* ou podem ser fornecidas pelo próprio usuário, através de uma lista de valores de freqüência, que são associadas aos valores distintos permitidos para o atributo.

Para geração dos dados foram construídos vários algoritmos, resultado da combinação das características de distribuição dos dados, disposição na relação, unicidade etc, que analisamos em seguida:

- *GenNumRandUniq*: gera uma distribuição de valores numéricos aleatórios únicos e com disposição aleatória na relação. Para um dado intervalo de valores máximos e mínimos, o algoritmo percorre seqüencialmente a coluna de valores aleatórios e seleciona aqueles que estão dentro do intervalo

de valores definidos pelo domínio do atributo. Como a coluna já se encontra com uma ordenação aleatória, a seqüência de valores obtidos também estará ordenada aleatoriamente. No caso deste algoritmo, para a geração dos valores utiliza-se a coluna de valores aleatórios permanente e, para a obtenção da disposição dos valores sobre as tupla, a coluna de valores aleatórios temporários. Para se garantir a integridade dos dados deve ser verificada a *1ª Regra*.

- *GenNumRandUnif*: obtém uma distribuição uniforme de valores numéricos aleatórios, com disposição também aleatória. Os valores obtidos, como no algoritmo anterior, são inseridos na relação em posições aleatórias obtidas da coluna temporária, com um número de vezes igual ao fator de repetição, FR. No caso do número de valores distintos (N) ser menor do que a cardinalidade da relação, $Card(R)$, então: $FR = Card(R)/N$ e caso contrário, $FR = 1$. Da mesma forma que no algoritmo anterior, não existe a possibilidade de inserção de valores distintos na mesma posição da relação, porque as posições são selecionadas seqüencialmente da coluna de valores aleatórios; como estes valores são garantidamente sem repetição, não corre esta possibilidade.

- *GenStrRandUniq*: distribui valores do tipo *string* com valores únicos, gerados aleatoriamente, com tamanho fixo ou variável, e disposição dos dados também aleatória. O formato dos atributos é aquele descrito na pagina 74. O valor numérico N, utilizado na 2ª parte, deve estar no intervalo entre $1..N_d$, onde N_d é o número de valores distintos do atributo. Se o atributo possuir uma disposição seqüencial, então o valor de N deverá ser seqüencial e caso a disposição seja aleatória, então N deverá ser obtido aleatóriamente dentro do intervalo já especificado. No caso de *chaves estrangeiras* o intervalo deve ser o mesmo intervalo da chave primária original e o nº de valores distintos deve ser igual à *Cardinalidade* da relação origem, de acordo com a regra de integridade da chave estrangeira. A 3ª parte é formada pela concatenação de caracteres não significativos, com o intuito apenas de formar o tamanho final do atributo. No caso do atributo ser de tamanho fixo, o valor da terceira parte ($TailSize$) é calculado por:

$$TailSize = Max - MiddleSize - 1$$

Onde:

$$MiddleSize = Comprimento(2ª parte)$$

Caso o atributo seja de tamanho variável então:

$$TailSize = GetNésimo(N) mod (Max - MiddleSize)$$

A função *GetNésimo*(N) obtém o elemento da coluna de valores aleatório únicos fixa que está na posição N.

Pode-se verificar, desta forma, que o algoritmo que constroi os atributos é determinístico. Ou seja, para um número de valores distintos N_d, os valores obtidos, tanto na ordem seqüencial quanto aleatória, e com tamanho fixo ou variável, serão sempre os mesmos.

- *GenStrRandUnif*: o algoritmo é basicamente igual ao *GenStrRandUniq*. Só que, neste caso, os valores são distribuidos uniformemente, ou seja, pode haver repetição de valores. O fator de repetição pode ser calculado da mesma forma que em *GenNumRandUnif*.

- *GenNumSeq*: gera um seqüência de valores numéricos que se inicia no valor mínimo definido no domínio do atributo; o valor máximo a ser obtido deve ser: $max = min + Card(R)$.

- *GenStrSeq*: gera uma seqüência de valores com uma ordem seqüencial. O formato dos dados é o mesmo utilizado para a geração de *strings* com disposição aleatória (pagina 74).

- *GenStrZipf*: este algoritmo gera valores do tipo *string* conforme uma distribuição do tipo *Zipf*. Os valores podem ser obtidos automaticamente pelo modelo a partir das descrições dos atributos ou, então, podem ser fornecidos pelo usuário em uma lista de valores. Para os valores obtidos automaticamente pela própria ferramenta, utiliza-se o mesmo algoritmo de *GenStrRandUnif*. Para obter-se a distribuição de *Zipf* as freqüências (ω_i) de cada elemento distinto são calculadas de acordo com 5.1 e inseridos na relação em posições aleatórias.

$$\omega_i = \frac{1}{(H_n \times rank_i)} \tag{5.1}$$

onde H_n é calculado por 2.2 e $rank$ é a posição do elemento em uma ordenação de acordo com a sua freqüência, onde $rank = 1$ para a posição do elemento mais freqüente e $rank = N_d$ a do menos freqüente.

- *GenAleatório*: obtém uma distribuição numérica aleatória sem qualquer controle de unicidade. Os valores gerados devem estar no intervalo entre $[min...max]$; e são calculados por $random(max - min) + min$. Neste caso não existe um controle da geração dos dados (não determinismo). Esta função somente deve ser utilizada quando o valor do atributo tem pouca influência sobre o desempenho, como, por exemplo, quando o atributo não é utilizado como argumento de nenhuma cláusula de seleção e, portanto, não existe a necessidade de controlar sua seletividade.

- *GenUser*: aqui o usuário pode especificar uma distribuição de dados diferente daquelas proporcionadas pela ferramenta. Para cada valor distinto do atributo deve ser informada uma freqüência associada. Ainda, os valores dos dados podem ser gerados automaticamente ou então podem ser fornecidos pelo próprio usuário.

5.5.3 Construção de chaves

Em nosso modelo de representação dos dados podemos construir *chaves simples* e *chaves compostas*. As *chaves simples* são obtidas com funções como *GenNumRandUniq* ou *GenStrRandUniq* para chaves numéricas e alfanuméricas, respectivamente.

A obtenção de *chaves compostas* utiliza uma solução um pouco diferente. Chaves compostas são combinações de mais de um atributo. Sendo assim, precisamos ter a certeza que não haverá repetição de nenhum dos valores gerados, ou seja, a composição de valores dos atributos que fazem parte da chave deve ser única.

Algoritmo para gerar chaves compostas

Chaves compostas são formadas por mais de um atributo. Em uma chave composta podem participar atributos de outras relações juntamente, ou não, com atributos da própria relação.

Estes atributos por sua vez devem formar subchaves próprias, quer dizer, devem ser chave primária de uma outra relação, estabelecendo-se, assim, uma regra recursiva, pois uma subchave de uma relação pode ser chave primária composta de outra relação, e assim por diante.

Para que exista integridade, todos os valores de subchaves, que pertençam a chave composta, devem ocorrer também na relação originária, da qual é chave primária.

O algoritmo para solução deste problema leva em consideração todas essas condições, resumindo:

- A chave composta pode ser formada somente por atributos de outras relações, ou com atributos de outras relações juntamente com algum atributo da própria relação.
- Atributos de outras relações são agrupados em subchaves próprias.
- Atributo da própria relação (pode existir apenas 1) que pertença à chave composta, é definido como uma subchave não-própria.
- O número máximo de valores distintos para um chave composta (*MAXKEYS*) deve ser igual ao produto das cardinalidades máximas de cada subchave própria e/ou não-própria.

- A cardinalidade de uma subchave é igual à cardinalidade da relação origem da qual ela é chave primária para o caso de uma subchave própria e é igual ao número de valores distintos definido no domínio do atributo, para o caso de uma subchave não-própria.
- A cardinalidade da relação que possui chave composta deve ser menor ou igual a *MAXKEYS* (regra de integridade).
- Qualquer valor de subchave própria deve existir na relação da qual a subchave é chave primária.

A seguir descrevemos o algoritmo de geração de chaves chaves compostas (*GenCompositeKeys*):

> Em primeiro lugar separam-se os atributos da relação que fazem parte da chave composta, agrupando os atributos de cada subchave. Em seguida calcula-se o *MAXKEYS* como foi descrito anteriormente. O numero máximo de chaves, definido por *MAXKEYS*, corresponde à cardinalidade do produto cartesiano dos domínios de todas as subchaves que fazem parte da chave composta. Estabelece-se, então, uma associação de um número no intervalo de: $1..MAXKEYS$, com uma combinação de subchaves que pertença ao produto cartesiano. O produto cartesiano é obtido através de uma combinação ordenada dos conjuntos de valores válidos para cada subchave (domínio da subchave). Esta combinação é produzida utilizando-se recursão, pois uma subchave pode ser uma chave primária composta de outra relação. A recursão termina quando se atinge uma subchave formada por um único atributo (final da recursão). Neste ponto, a subchave pode ser um atributo da própria relação (sub-chave não própria), ou então uma chave estrangeira (chave primária de outra relação). De qualquer forma, é possível identificar o seu domínio e obter o valor da subchave para o número da combinação desejada. O número escolhido no intervalo: $1..MAXKEYS$, associado à combinação, pode ser convertido em uma posição relativa dentro do domínio de uma subchave, ou seja, para um dado valor, a posição relativa dentro do domínio da primeira subchave, conforme a ordernação das subchaves, é calculada pelo resto da divisão inteira (módulo) da posição inicial pela cardinalidade da subchave. Para as subchaves seguintes a posição original é dividida seguidamente, quando se passa de uma para outra subchave, pela divisão inteira da posição pela cardinalidade de cada subchave e dentro de cada domínio obtem-se o valor desejado fazendo-se a divisão por módulo.

Desta forma, o algoritmo pode ser utilizado de duas maneiras diferentes: as posições podem ser valores sequenciais ou podem ser valores aleatórios únicos,

dentro do intervalo de posições válidas. Neste último caso, a única modificação no algoritmo é a inclusão de um nível de indireção na obtenção do valor da subchave dentro de seu domínio.

5.6 Modelo de carga de trabalho

As transações são obtidas através da utilização de um *Modelo de Carga de Trabalho* que é composto de um *Modelo de Descrição de Transações* e um *Modelo de Execução*. O *Modelo de Descrição* define o conteúdo das transações que são geradas em uma seqüência definida pelo *Modelo de Execução*.

5.6.1 Modelo de descrição de transações

O *Modelo de Descrição de Transações* utiliza um paradigma de manipulação de bases de dados através de janelas (formulários). Neste enfoque as transações são desenvolvidas centradas em uma visão base sobre a qual são realizadas as interações, e outras operações (seleções, atualizações, deleções) são realizadas em outras tabelas relacionadas à visão base, como foi discutido na página 60 na Seção 4.6.1.

Descrição das transações

No *Modelo de Descrição* são representadas dois tipos de transações: *consulta* e *inclusão*. As transações de consulta podem ser seguidas de atualizações ou remoções, conforme as restrições impostas à visão associada à janela. As transações de consulta baseiam-se na estrutura de uma *consulta genérica* que será apresentada adiante.

Transações de consulta são traduzidas em SQL para comandos SELECT. As atualizações ou deleções, caso ocorram, serão traduzidas para comandos UPDATE e DELETE, respectivamente. A condição de seleção, representada pela cláusula WHERE, dos comandos UPDATE e DELETE, deverão ser as mesmas do comando SELECT, caso a consulta seja seguida de atualização ou remoção.

A identificação dos atributos que são recuperados em uma consulta, do ponto de vista de desempenho do SGBD, não é importante. O otimizador de transações leva em consideração apenas aqueles atributos que participam da cláusula WHERE, que são utilizados para se determinar a escolha do caminho de acesso, baseado na existência de índices ou *agrupamentos* por exemplo.

As cláusulas de condição (WHERE) são formadas por predicados; cada predicado possui uma seletividade associada. Baseando-se em hipóteses teóricas de uniformidade [CHRI84], podemos calcular a seletividade dos seguintes tipos predicados:

$$c < v \quad s = (v - min)/(max - min) \tag{5.2}$$

$$c > v \quad s = (max - v)/(max - min) \tag{5.3}$$

$$c = v \quad s = 1/N_{distintos} \tag{5.4}$$

$$v_1 \leq c \leq v_2 \quad s = (v_2 - v_1)/(max - min) \tag{5.5}$$

Nestes cálculos leva-se em consideração a distribuição de valores uniforme sobre o atributo c. Os valores *max* e *min* definem o domínio dos valores do atributo, no caso de valores numéricos.

Nas combinações de predicados assumimos independência dos valores dos atributos. No caso de serem dados dois predicados P_1 e P_2 com seletividades S_1 e S_2, respectivamente:

$$P_1 \ and \ P_2 = S_1 \times S_2 \tag{5.6}$$

$$P_1 \ or \ P_2 = S_1 + S_2 - S_1 \times S_2 \tag{5.7}$$

Consulta genérica

Definimos uma *consulta genérica* estruturada em três níveis diferenciados de abstração (Figura 5.1). O primeiro nível é representado pela materialização da

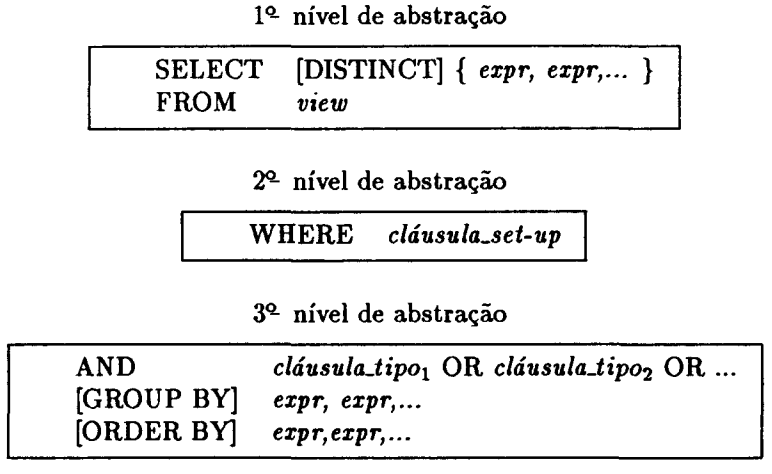

Figura 5.1: Estrutura da consulta genérica

visão associada à janela corrente. Devido à própria facilidade de utilização do paradigma de janelas, a tarefa de definição do modelo de transações torna-se

bastante beneficiada. Usualmente, as aplicação desenvolvidas neste paradigma possuem janelas, que consideramos como janelas mestras (ou *janelas principais*), e janelas de detalhe (ou *secundárias*). As *janelas principais* têm a incumbência de mostrar informações correspondentes aos níveis superiores de uma estrutura hierárquica. As *janelas secundárias* fornecem informações mais detalhadas a respeito das informações obtidas nas janelas principais correspondentes (como exemplo do uso do paradigma, veja a explicação da figura 4.2 na página 66). Para isso, importam dados para realizar parte da seleção.

O segundo nível de abstração corresponde à seleção tomando-se valores importados de outras janelas. Utilizamos neste caso conjunções de predicados do tipo: $c = v$, que correspondem a generalização das janelas secundárias.

O terceiro nível de abstração corresponde a diversos tipos de seleção dentro de uma mesma janela. Como pode existir mais de um tipo de consulta por janela, obtemos uma disjunção de predicados do tipo: $c_1 = v_1$ *or* $c_2 = v_2$ *or* ...

.

A cláusula *cláusula_set-up* somente aparece em janelas secundárias, onde seu valor deve ser o mesmo daquele obtido na construção da janela principal.O formato do predicado *cláusula_tipo*$_i$ é $c_1 = v_1$ *and* $c_2 = v_2$ *and*... .

Nos segundo e terceiro níveis deve-se, ainda, verificar a seguinte restrição:

$$Seletividade \times Cardinalidade \geq 1 \tag{5.8}$$

A seletividade pode ser calculada pelas equações de 5.2 a 5.5, se considerarmos distribuições uniformes sobre os atributos.

Para as transações de INCLUSÃO e ATUALIZAÇÃO devemos verificar as seguintes restrições:

- Apenas em visões formadas de uma relação base.
- Atributos gerados através de inclusões ou atualizações que sejam chaves estrangeiras, devem possuir o valor já gerado anteriormente na relação primária.
- Chaves simples e compostas devem ter a unicidade preservada.
- Distribuição dos valores e disposição dos dados devem ser preservados.

A atualização, mostrada na figura 5.2 é composta de uma seleção seguida da atualização propriamente.

Os valores utilizados na atualização serão os mesmos daqueles obtidos pela seleção. Do ponto de vista de desempenho, é indiferente se os valores são os próprios valores lidos ou algum outro. Por outro lado, ganhamos na simplificação do modelo.

De forma semelhante, a estrutura da remoção genérica é definida na figura 5.3, que mostra apenas o comando de remoção. Neste caso o formato da consulta é o mesmo definido para a atualização.

Geração dos dados das transações

As funções de geração dos valores para os atributos foram utilizadas inicialmente para carregar as relações. Dentro do Modelo de Carga de Trabalho, estas mesmas funções voltam a ser utilizadas, mas, neste caso, elas serão utilizadas de modo diferente. Existem duas situações distintas de utilização desses algoritmos: na obtenção de valores previamente gerados, para o caso de uma consulta, e na obtenção de novos valores, para o caso de uma inclusão. Em qualquer dos casos, as restrições de integridade definidas pelas regras de integridade e a manutenção do domínio dos atributos devem ser mantidas, valores máximos e mínimos preservados, unicidade de chaves etc. Para que isso ocorra, cada função deve atender algumas restrições:

- *GenNumRandUniq*: esta função fornece uma distribuição de dados numéricos uniforme, os valores obtidos são únicos e a disposição dos dados é aleatória. Para que um novo valor seja gerado, a restrição de unicidade, Regra 4.1, deve ser novamente verificada.

  Após a geração inicial das relações, salva-se a última posição utilizada da coluna de valores aleatórios únicos.Para os novos valores a serem gerados subseqüentemente, reinicia-se a partir da última posição utilizada, que deve ser atualizada para cada novo valor a ser gerado.

- *GenNumRandUnif*: fornece uma distribuição uniforme de dados numéricos, com disposição aleatória sobre as tuplas. Como não existe a preocupação de unicidade, os valores podem ser obtidos por uma função de geração de números aleatórios: $random(min, max)$. Neste caso a criação dos novos valores não é determinística, a última posição na coluna de valores aleatórios deverá ser mantida e não deve ser alterada após a carga inicial das relações.

- *GenStrRandUniq*: a partir da última posição utilizada da coluna de valores aleatórios únicos inicia-se a geração de novos valores. Para se obter valores que já existam nas relações, basta obter aleatoriamente um valor da coluna de números aleatorios únicos, no intervalo entre: $1..N_d$, onde a seleção (busca circular) é realizada da primeira posição da coluna até a última posição utilizada na carga inicial.

- *GenStrRandUnif*: o funcionamento é semelhante a geração de um valor já inserido na relação. Para isso, escolhe-se uma posição na coluna de valores aleatórios únicos entre 0 e a última posição utilizada. Este valor é utilizado para construir o *string* desejado, conforme suas características.

- *GenNumSeq*: obtém-se o último valor criado, e reinicia-se a partir dai.

- *GenStrSeq*: o seu funcionamento é o mesmo do anterior.

- *GenStrZipf*: após a carga inicial das relações, o atributo deve possuir, neste caso, uma distribuição tipo Zipf com N valores distintos. Esta distribuição é caracterizada pela freqüência (w_i ; $i = 1..N$) de ocorrência de cada elemento. Para obtermos um valor para inclusão ou para consulta devemos respeitar a distribuição de Zipf, ou seja, alguns valores são mais referenciados do que outros. Para isso, a obtenção dos valores deve levar em conta a freqüência de ocorrência da distribuição Zipf e assim manter estes valores para novas inclusões na relação. Para que isso ocorra, utilizamos o algoritmo apresentado na figura 5.4 onde a função *ObtemFreqZipf* calcula a freqüência do iésimo elemento do domínio do atributo, conforme a equação 5.1, e a função *TransfZipf* gera um string de acordo com as características desejadas.

- *GenCompositeKeys*: se a geração for seqüencial, reinicia a partir do último valor de cardinalidade da relação. Para disposição aleatória, obtém-se o último valor aleatório utilizado e reinicia, a partir dai, com o mesmo algoritmo já definido anteriormente.

- *GenAleatório*: como não há a preocupação com valores aleatórios únicos, a geração prossegue como anteriormente.

5.6.2 Modelo de execução de transações

Na primeira parte do *script* do *benchmark* realizou-se a criação dos objetos do banco de dados e a geração de valores. Agora, segue-se a geração da carga de trabalho. Nesta parte, o *Modelo de Descrição de Transações* é utilizado para obter-se as transações que serão geradas segundo uma seqüência definida pelo *Modelo de Execução de Transações*.

O *Modelo de Execução* possui uma estrutura hierárquica, onde procuram-se representar dois níveis diferenciados de concorrência de execução das transações:

- *A Nível do Ambiente*: diferentes aplicações podem estar sendo executadas concorrentemente no SGBD.

- *A Nível da Aplicação*: mais de um usuário pode utilizar a mesma aplicação simultaneamente.

A partir desta configuração hierárquica e utilizando o *modelo genérico de consultas* já descrito, foi construído um *driver* que percorre a estrutura, gerando uma mistura de transações que procura representar o modelo fornecido.

Ambiente

O *Modelo de Execução* é estruturado hierarquicamente, começando com a descrição do *ambiente*. O ambiente pode ser *mono* ou *multi-tarefa*. Em um ambiente mono-tarefa, apenas uma aplicação está sendo executada e, em um ambi-

ente multi-tarefa, mais de uma aplicação pode estar sendo executada simultaneamente no SGBD (Figura 5.5) . Para cada ambiente o usuário deve especificar:

- Número de Aplicações Concorrentes ($N_{conc.}$).
- Período de observação (P).
- Número Máximo de Aplicações para o ambiente (N_a).
- Freqüência da Aplicação (f_a).

O somatório das freqüências de utilização de cada aplicação f_{a_i} devem verificar: $\sum_{i=1}^{N_{conc.}} f_{a_i} = 1$.

Para a geração do *script* da carga de trabalho, o *driver* deve escolher aleatoriamente uma das aplicações para ser executada. Se o ambiente for mono-tarefa, escolhida uma aplicação, deve-se executar o número total de transações configurada para a aplicação escolhida. No caso do ambiente ser multi-tarefa, podem ser executadas transações de diferentes aplicações de maneira intercalada. Para cada aplicação deve ser calculado o número total de utilizações dentro do período de observação (P). Para cada aplicação este valor é obtido por $f_{a_i} \times P$. Quando não houver mais seções na aplicação (mono-tarefa) ou transações para serem executadas (multi-tarefa), termina a geração da carga de trabalho para o ambiente.

Como se pode observar, no caso do ambiente multi-tarefa o nível de embaralhamento das transações é mais elevado do que no caso mono-tarefa. Este fato pode influenciar sobremaneira o desempenho do SGBD, com ênfase nas políticas de gerência de *buffer-pool.*

Aplicação

Uma aplicação é representada no modelo através da composição de várias janelas. A figura 5.6 mostra o *modelo de execução de uma aplicação.* Neste modelo o usuário deve informar:

- Número total de janelas na aplicação (N_a^j).
- Número de usuários concorrentes ($N_{usu.}$).
- Número de seções na janela / aplicação (N_a^s).
- Número de transações na janela / seção (N_s^t).

Uma aplicação pode ser executada em modo mono-usuário ou em modo multi-usuário. No caso mono-usuário o *driver* simula a utilização da aplicação por apenas um usuário. Assim, o *driver* escolhe aleatoriamente uma janela para executar e permanece na janela escolhida até que sejam executados todas as transações possíveis. Cada janela, por sua vez, pode ser escolhida até N_a^s

vezes. No caso multi-usuário o driver simula a ocorrência de transações de janelas possivelmente diferentes, como se elas estivessem sendo usadas ao mesmo tempo. Quando não existir mais janelas para executar, com o N_a^s já esgotado para o caso mono-usuário, ou quando o número total de transações por janela $N_s^t \times N_a^s$ (caso multi-usuário) já tiver sido realizado, o *driver* termina a execução da aplicação escolhida.

Janela

Uma janela pode executar transações de dois tipos: consulta e inclusão; a estrutura do *modelo de execução de uma janela* é mostrada na figura 5.7. Para uma dada janela corrente, o *driver* escolhe aleatoriamente uma transação (consulta ou inclusão). Dentro de uma mesma janela podem ser executados até N_s^t transações. O número de transações de consulta é fornecido por $N_{sel.} = N_s^t \times f_{sel.}$, e o número de inclusões é dado por $N_s^t - N_{sel.}$. Neste caso o usuário deve especificar a freqüência de consultas ($f_{sel.}$).

Consulta

No *modelo de execução de consulta* podem existir várias consultas diferentes para a mesma janela, ou seja, o usuário pode especificar cláusulas de seleção diferentes (ver o modelo de consulta genérica na pag. 80). A estrutura de uma consulta é mostrada na figura 5.8. Para cada consulta específica temos uma freqüência parcial ($f_{parc.}$). O somatório das freqüências parciais devem verificar:

$$\sum_{i=1}^{NTC} f_{parc._i} = 1$$

Onde NTC é o número total de transações de consulta. Neste modelo, as consultas podem ser de três tipos:

- Consultas simples
- Consultas com atualização
- Consultas com remoção

Para as consultas com atualização, deve ser informada a freqüência de atualização ($f_{atualiz.}$), assim como, no caso de haver remoção, a freqüência respectiva ($f_{rem.}$). Desta forma, fornecidas as configurações de consultas para uma janela, o *driver* pode calcular o número total de consultas para cada tipo específico. Para uma consulta i, o número total de execuções (N_{total}) é dado por:

$$N_{total} = N_t^s \times f_{sel.} \times f_{parcial_i}$$

Para calcular o número de consultas com atualização de uma consulta C_i temos:

$$N_{atualiz.i} = N_{total} \times f_{atualiz.i}$$

e o número de remoções é dado por:

$$N_{rem.i} = N_{total} \times f_{rem.i}$$

e o número de consultas simples é:

$$N_{cons.i} = N_{total} - N_{atualiz.i} - N_{rem.i}$$

Cada tipo de consulta de uma janela equivale a uma cláusula de seleção diferente (*cláusula_tipo*), de acordo com o modelo de consulta genérico mostrado na figura 5.1.

5.7 Ambiente de execução e testes

Para o desenvolvimento do *benchmark* o ambiente disponível foi:

- microcomputador PC compativel (Intel 80286)
- sistema operacional MS-DOS
- SGBD ORACLE versão 4
- Ferramentas para banco de dados Pascal Toolbox

O ambiente de execução do *benchmark*, SGBD *ORACLE* versão 4, trouxe alguns problemas de implementação. O ideal seria utilizar a técnica de CURSOR [DIMM86], onde se poderia ter acesso às bases de dados internas ao SGBD em utilização (em nosso caso o ORACLE) durante a execução da ferramenta. Desta forma, poder-se-ia construir tabelas, inserir valores e consultar as tabelas já criadas com mais facilidade. Isso no entanto não foi possível, pois não contávamos com o precompilador para linguagem C ou Pascal (Pro*C ou Pro*Pascal), que permitiria a inclusão de comandos SQL diretamente no programa fonte. Sendo assim, a solução adotada foi realizar o *benchmark* em modo *batch*, ou seja, a ferramenta gera um arquivo texto com todas as operações necessárias (definição da relações, views, inclusão de valores, carga de trabalho etc.) que, em seguida, são executadas pelo SGBD.

SELECT	[DISTINCT] { *expr, expr,...* }
FROM	*view*
WHERE	*cláusula_set-up*
AND	*cláusula_tipo*$_1$ OR *cláusula_tipo*$_2$ OR ...
[GROUP BY]	{ *expr, expr,...* }
[ORDER BY]	{ *expr,expr,...* }
FOR UPDATE OF	*atributo, atributo, ...*

UPDATE	view
SET	$atributo_1 = atributo_1$, $atributo_2 = atributo_2$
WHERE	*cláusula_set-up*
AND	*cláusula_tipo*$_1$ OR *cláusula_tipo*$_2$...OR *cláusula_tipo*$_n$

Figura 5.2: Estrutura da atualização genérica

DELETE FROM	view
WHERE	*cláusula_set-up*
AND	*cláusula_tipo*$_1$ OR *cláusula_tipo*$_2$...

Figura 5.3: Estrutura da remoção genérica

```
procedure GeraZipf(
    N         : longint; { No. valores distintos        }
    Cr        : longint; { Cardinalidade relacao        }
    size      :  integer;{ Tamanho do atributo          }
    formato   : string;  { formato do atributo
var value     : string );{ Valor retornado p/ o elemento}

var
    w         : real     { frequencia do elemento }
    i,                   { identifica o rank }
    v,                   { valor de teste }
    inf,                 { limite inferior }
    sup,                 { limite superior }
    delta     : longint { no. de valores no intervalo }
    inside    : boolean { false se valor fora do intervalo }
  begin

    { Inicializacao }

    v:= random(Cr) + 1; { obtem um valor aleatorio }
    inf:= 1;
    sup:= 0;
    inside:= false; i:= 1;
    { Encontra o elemento distinto correspondente }
    while (i <= N) and (not inside) do
      begin
        ObtemFreqZipf(w,i,N); {Calcula a freq. w para o elemento i}
        delta:= round(w * Cr);
        sup:= sup + delta;
        if (v >= inf) and (v <= sup) then
            inside:= true
        else
            i:= i+1;
        inf:= inf + delta
      end;

    {gera string no formato desejado}

    value:= TransfZipf(i,size,formato)

end;            {        GeraZipf        }
```

Figura 5.4: Algoritmo para obtenção de uma distribuição tipo Zipf.

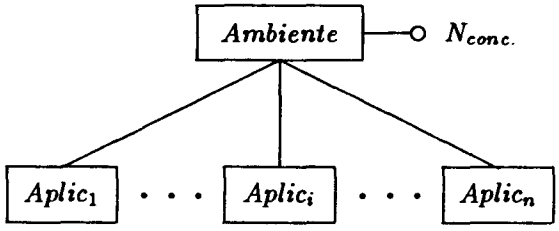

Figura 5.5: Modelo de execução do ambiente

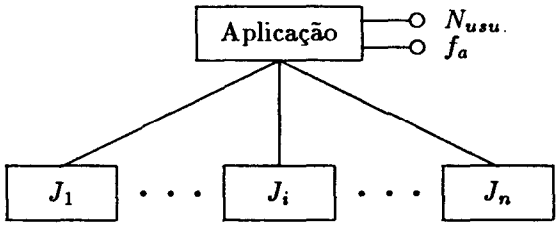

Figura 5.6: Modelo de execução da aplicação

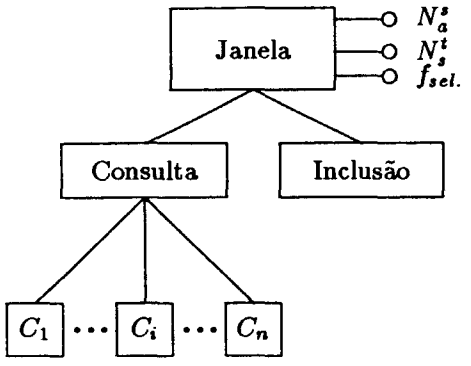

Figura 5.7: Modelo de execução da janela

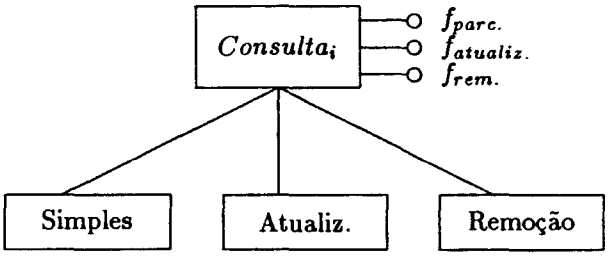

Figura 5.8: Modelo de execução de uma consulta

Capítulo 6

Avaliação da metodologia

A avaliação da implementação da ferramenta foi realizada tomando-se duas aplicações diferentes. Na primeira aplicação utilizamos o *Benchmark de Wisconsin* (*BW*) e verificamos como pode ser obtido pela ferramenta. Na segunda aplicação fizemos um estudo de caso para um sistema existente, que faz parte do Projeto SAGRE. Neste estudo, verificamos a facilidade de utilização da metodologia e a sua representatividade. Mas, antes de passar para a avaliação, propriamente, faremos alguns comentários sobre a aplicação da metodologia em bancos de dados não relacionais.

6.1 Aplicação para bancos de dados não relacionais

O Modelo de Carga de Trabalho da metodologia é baseado no conceito de visão de banco de dados, tal como é definida para o modelo relacional, apesar de alguns SGBDs não possuirem esta facilidade. O conceito de visão, na metodologia apresentada, é bastante importante, pois permite a definição de janelas de consulta ao banco de dados de complexidade razoável. Ao mesmo tempo, simplifica a construção das transações [1] para estas janelas. Isto torna-se possível devido à abstração proporcionada pelo conceito de visão.

A aplicação desta metodologia deverá ser adaptada para SGBDs que não possuem o mecanismo de definição de visões. Ao invés de se utilizar uma visão, substitui-se esta definição por um procedimento escrito em alguma linguagem de programação disponível para o SGBD em teste, realizando-se, assim, a mesma função que seria realizada pela visão. Neste caso, a metodologia perderia um pouco a generalidade, pois, existiria uma versão diferente do conjunto de procedimentos necessários para cada SGBD.

[1] Utilizando para isso a estrutura de uma consulta genérica, como foi mostrado no capítulo anterior

6.2 Validação pelo BW

Ao analisarmos a implementação da ferramenta, utilizando o *BW*, verificamos que a metodologia é, no mínimo, representativa deste *benchmark*, visto que ele pode ser gerado pela ferramenta.

O *benchmark* de Wisconsin, tal com descrito em [BITT89], é formado por apenas 3 relações base: onek com 1000 tuplas, tenk1 e tenk2, ambas com 10.000 tuplas e a mesma estrutura (esquema).

Inicialmente as relações são construídas a partir de suas descrições. Pode-se verificar sem maiores problemas que a construção das relações é bastante simples pois a estrutura das relações, a descrição dos atributos, distribuição dos dados e disposição das tuplas sobre a relação podem ser totalmente representadas no Modelo de Representação dos Dados adotado pela metodologia.

Atributos como *unique1* e *unique2* são chaves e possuem respectivamente disposição aleatória e seqüencial. Da mesma forma, *stringu1* e *stringu2* também possuem disposição aleatória e seqüencial, sendo que ambos também são chaves da relação.

As transações utilizadas no *BW* são representadas pela metodologia sem maiores problemas. Neste sentido, o modelo é bastante simples: existem 8 visões associadas às 8 janelas que compõem, o que supostamente seria, a aplicação; na realidade, o BW não procura representar uma aplicação real.

A maior parte das janelas, assim obtidas, possuem pouca variedade de transações, sendo a maior parte delas composta de seleções de um apenas um tipo. Somente uma janela possui uma variedade maior de transações, com seleções, atualizações, remoções e modificações.

As consultas às visões associadas são, na maior parte das vezes, seleções simples e apenas uma janela possui consultas com projeções (opção DISTINCT) e funções sobre agregados (MIN e SUM). O resultado completo da configuração do *BW* se encontra no Apêndice A.

6.3 Aplicação em uma situação real

Para a validação da segunda parte, escolhemos uma aplicação que está em desenvolvimento por empresas do Sistema Telebrás, o *Sistema Automatizado de Gerência da Rede Externa (SAGRE).*

6.3.1 Descrição

De uma forma simplificada, chama-se *rede externa* (*outside plant*), o conjunto de cabos e fios telefônicos de distribuição externa, equipamentos e acessórios externos às estações telefônicas e estruturas de suporte (postes, caixas e canalizações subterrâneas), destinadas a interligar os telefones às estações, bem como estas entre si [SAGR90].

A *Gerência de Rede Externa* engloba o conjunto de atividades técnicas que visam otimizar a utilização das facilidades de rede externa existentes e prover novas facilidades nos lugares onde elas são necessárias, de forma a atender as solicitações dos assinantes e usuários das empresas. As atividades de gerência de rede externa são as seguintes:

- Levantamento e Estudo de Demanda
- Planejamento Técnico
- Projeto de Rede
- Implantação de Rede
- Operação e Manutenção
- Controle de Contrato

Geralmente as atividades ligadas à administração de rede externa ficam subordinadas aos departamentos técnicos (Engenharia, Operações) das empresas de telecomunicações que, normalmente, já possuem em funcionamento vários sistemas automatizados de controle (sistemas de cálculo de demanda, controle de materiais, atendimentos a problemas de assinantes, etc), que compartilham um grande volume de informações com o SAGRE. Além disso, esses sistemas possuem sérios problemas de integração, ocorrendo uma redundância de dados indesejável.

Atualmente, as tarefas de planejamento, projeto, implantação e operação são muito dificultadas devido à falta de informações precisas e atualizadas sobre as plantas da rede.

O instrumento básico de um projetista são as plantas cadastrais que informam a situação atual, as previsões de demanda etc. No entanto, essas plantas estão constantemente desatualizadas.

Em uma empresa de grande porte como a Telesp, por exemplo, são necessárias milhares de plantas, que, muitas vezes, possuem mais de uma versão de uma mesma planta espalhada em diferentes setores da empresa (operação, implantação e projeto). Não muito raramente, em todo trabalho a ser iniciado, se exige uma atualização quase completa das informações contidas nas plantas, pois as diferenças entre as várias versões existentes pode ser bastante grande. Em muitos casos, é necessário que essa atualização seja feita em campo, devido a completa falta de confiabilidade dos registros existentes.

Tais problemas, em boa parte, decorrem da demora na atualização das plantas. Como, na maioria das vezes, elas ainda são feitas em papel vegetal e nanquim, na medida em que são realizadas as alterações, chega-se a um ponto onde não existe mais a possibilidade de ser corrigida a planta antiga, e uma nova planta deve ser refeita, encarencendo mais ainda o processo.

O *SAGRE* deverá proporcionar um ambiente que facilite a execução das funções de administração de rede. Para isso, ele deverá proporcionar a integração das informações entre os sistemas já existentes e oferecer recursos de computação gráfica para mapeamento geográfico, provendo, assim, informações cadastrais, alfanuméricas ou gráficas das plantas da rede, para as tarefas de planejamento, demanda, projeto, operação etc.

Também está previsto a utilização de estações de trabalho com capacidade de processamento numérico e gráfico, interligadas entre si por redes locais, e conectadas à computadores de grande porte, onde estão armazenados a maior parte dos dados, atualmente.

O ambiente de *software* proposto deverá utilizar ferramentas do tipo CAD [2], GIS [3] e bancos de dados distribuídos.

Até o presente momento já foram realizados levantamentos iniciais, estudos de viabiblidade técnica e obtido uma modelagem inicial do sistema utilizando MER e DFD.

Para efeito da utilização de uma aplicação na MBE, apenas uma parte do SAGRE foi escolhida para ser configurada no *benchmark*. As relações e descrição dos atributos que foram utilizadas se encontram no Apêndice C assim como a descrição e a configuração das transações.

A parte utilizada do SAGRE representa o *subsistema de rede de cabos* (cabos, emendas, caixas terminais, estações etc.) e o *subsistema de infra-estrutura subterrânea* (rede de dutos e caixas subterrâneas.) e aérea (postes). A escolha destes subsistemas é bastante natural pois, basicamente, o interesse central do sistema é oferecer controle sobre essas informações, através de mapas, plantas e de um banco de dados associado.

A aplicação da metodologia se inicia com a leitura das informações presentes no MER. Nesta etapa são fornecidos dados para:

- definição das relações.
- definição de atributos e domínios
- definição de chaves primárias (simples e compostas)
- chaves estrangeiras
- distribuição dos valores dos atributos, disposição nas tuplas da relação.

Uma dificuldade encontrada na etapa de representação da aplicação com suas relações e a subsequente geração dos dados, ocorreu na geração de chaves compostas quando foi encontrado um esquema do tipo figura 6.1. Neste caso, um possível mapeamento (somente mostrando a configuração das chaves primárias das relações) para o modelo relacional seria: a relação *estação* teria como chave

[2]CAD. Computer Aided Design.
[3]GIS. Geographical Information System.

num_estação, *cabo* teria *num_estação* composta com *num_cabo* e *lance_cabo* teria *num_estação* composta com *cod_lance_cabo*.

O relacionamento entre *cabo* e *lance_cabo* poderia possuir os atributos *num_estação*, *num_cabo* e *cod_lance_cabo*, pois o relacionamento é de muitos-para-muitos, ou seja, um lance de cabo pode possuir vários cabos assim como um cabo saindo da estação pode ser particionado em vários lances.

Como neste caso, assumimos que a chave primária de *lance_de_cabo* é formada pelos atributos: num_estação e cod_lance, quer dizer, o código do lance depende da estação ao qual está cadastrado. A dificuldade do modelo reside na obtenção dos valores para o relacionamento (chave composta).

Na realidade, o relacionamento entre as duas entidades deveria conter as chaves primárias de cada uma delas. Neste caso, as chaves primárias são chaves compostas e compartilham um atributo em comum (*num_estação*). Para se evitar a repetição deste atributo comum, ficariamos apenas com *num_estação*, *num_cabo* e *cod_lance_cabo*.

Além disso, somente faria sentido, do ponto de vista de integridade, se as tuplas geradas no relacionamento possuissem subchaves cujos valores existíssem nas relações de origem e, nas quais, os valores de *num_estação*, fossem os mesmos nas duas relações. Isso quer dizer que, para cada tupla (*num_estação*, *num_cabo*, *cod_lance_cabo*) deveria existir na relação *cabo* uma ocorrência de (*num_estação*, *num_cabo*) e em *lance_cabo* uma ocorrência de (*num_estação*, *cod_cabo*) com o mesmo *num_estação*. Ao contrário, estaríamos relacionando ocorrências das entidades que não se relacionam no MER, ou seja, não faria sentido um lance de cabo de uma estação relacionar-se com um cabo de outra estação.

Este problema poderia se contornado se incluíssemos cláusulas de restrição na configuração de um relacionamento. No entanto, esta solução complicaria por demais a representação dos dados, considerando-se o objetivo inicial da metodologia, que previa uma utilização simplificada na qual o usuário não precisaria preocupar-se com detalhes desse nível.

A solução adotada para esse problema foi modificar o esquema, tornando a entidade, neste caso, *lance_de_cabo*, existêncialmente independente (entidade forte) de *estação*. Assim, pode-se, inclusive, relacionar mais de uma estação com o mesmo lance de cabo. O que não deixa de refletir uma situação real, como na ligação entre estações com cabos tronco.

Situações como esta também acontecem quando uma entidade se relaciona com outras entidades através de um mesmo atributo. Por exemplo, quando se deseja localizar uma emenda de um cabo[4], ela pode estar fisicamente localizada em um poste (rede aérea) ou em uma caixa subterrânea (rede dutos). Na geração dos dados para estes dois relacionamentos não há como se evitar que a mesma emenda possa aparecer em um poste e simultaneamente em uma caixa subterrânea.

[4]Conexão entre as extremidades de dois ou mais lances de cabo, ou a um ponto terminal (caixa etc.)

6.3.2 Avaliação da implementação

A implementação foi realizada em Turbo-Pascal 5.0 [PASC87] com apoio de ferramentas para banco de dados, Database ToolBox [TOOL85], que implementam o dicionário de dados (modelo de representação dos dados e modelo de carga de trabalho) em arquivos do sistema operacional. Os ambientes de testes utilizados foram o SGBD ORACLE versão 4 para MS-DOS e micro-computador compatível com IBM-PC XT e ORACLE versão 5 para ambiente VMS/DEC-VAX.

Inicialmente a configuração do modelo de representação dos dados (relações, atributos, domínios etc.) e o modelo de carga de trabalho são transcritas em *arquivos de configuração* (arquivos .INI), que são carregados em seguida para o dicionário de dados da ferramenta.

Em seguida, estas informações são utilizadas para gerar *arquivos de trabalho*, DADOS.SQL e CARGA.SQL, que correspondem à carga inicial que gera o banco de dados da aplicação e à carga de trabalho da aplicação, respectivamente. Para a realização dos testes, estes arquivos são posteriormente executados dentro da interface SQL interpretado, disponível no SGBD (programa *ufi* para Oracle versão 4 e *sqlplus* para Oracle versão 5).

A geração do banco de dados no SGBD mostrou-se muito onerosa, devido a dois aspectos:

- *Elevada ocupação de espaço físico* em disco por arquivos de trabalho. Isso se fazia notar principalmente no caso da geração de um banco de dados muito grande (cardinalidade das relações elevada), que resulta em arquivos de trabalho muito extensos.

- *Elevado consumo de tempo* durante o carregamento do banco de dados no SGBD, pelo fato do carregamento ser realizado em SQL.

O problema da criação de arquivos de trabalho muito grandes se deve ao fato da geração dos dados ser realizada em duas fases: primeiro, a geração de um arquivo em SQL, para depois ser carregado no SGBD. Este problema poderia ser evitado se a geração dos dados fosse realizada diretamente no SGBD. Para isso, bastaria que o programa, que implementa a ferramenta, tivesse comunicação com o SGBD. Seria melhor, inclusive, se o dicionário de dados da ferramenta pudesse ser implementado no próprio SGBD, o que diminuiria, em muito, a complexidade do programa. Para que isso fosse possível precisaríamos dispor de um pré-compilador (Pro*C ou Pro*Pascal) [DIMM86], que permitiria a comunicação direta com o SGBD via *SQL embutido*. Como não dispunhamos do precompilador no ambiente de desenvolvimento, esta hipótese não foi levada a frente.

Na realidade, dispunhamos de um precompilador no ambiente VAX-VMS/DEC, que não foi utilizado, no entanto, pelo fato de nosso acesso a este sistema ser muito precário. O acesso era realizado por uma ligação entre um micro, localizado em Brasília, e o VAX localizado no CPqD em Campinas, utilizando-se

um modem com linha discada a 1200 full-duplex, protocolo *tty* e conversão para X-25 (Serviço Renpac) para o acesso ao VAX. Devido à baixa velocidade de transmissão, que implicaria em um tempo muito longo entre a transferência do programa para o VAX, a compilação, testes, e correção no microcomputador em casos de erros, a alternativa de utilizar-se o precompilador, apesar de ser bastante interessante, foi abandonada.

Quanto ao problema do tempo dispendido na criação das tabelas do *benchmark*, optamos pela escolha do carregamento via SQL, que apesar de ser mais lenta, não apresentaria problemas de compatibilidade com outros SGBDs. Uma possível alternativa, caso não se desejasse utilizar SQL, seria a criação de um arquivo com um formato específico de algum utilitário que fizesse o carregamento, como por exemplo o ODL[5] [DIMM86].

Além disso, mesmo optando pelo carregamento via SQL, poderíamos realizá-lo ainda de forma mais rápida. Na forma como foi gerado o arquivo de trabalho, DADOS.SQL, cada atributo é gerado um de cada vez, forçando o sistema a carregar cada relação, atributo por atributo, ou seja, o sistema percorre a relação um número de vezes igual ao número de atributos que ela contém (apenas no caso de chaves compostas isso não ocorre). Este processo é muito mais demorado do que se pudéssemos carregar a relação por uma linha completa a cada vez, evitando-se percorre-la completamente para cada atributo. Por outro lado, a opção de carregar um atributo de cada vez nos pareceu bastante natural, pois os algoritmos de geração dos valores são orientados para atributos, individualmente. Para que pudessemos carregar uma linha de cada vez, com todos os valores dos atributos gerados previamente, em primeiro lugar necessitaríamos gerar os dados individualmente, armazená-los na memória do computador, montar a relação desejada na memória e, finalmente, gerar comandos INSERTs que incluiriam todos os atributos de uma só vez.

6.3.3 Resultados obtidos

Apresentamos na tabela 6.1 os resultados para a execução do *benchmark SAGRE* em uma máquina VAX 785, utilizando o SGBD Oracle. Os resultados mostram uma média dos valores de: tempo total de execução, tempo de CPU, número de operações E/S e operações de paginação. Os valores foram obtidos através do comando "SET TIMING ON", disponível no SGBD; um extrato dessa execução pode ser encontrado no Apêndice D.

Com os resultados da execução do *benchmark*, podemos verificar o desempenho de algumas alternativas, como, por exemplo, a criação de índices e verificar o efeito resultante sobre o desempenho. Ao criarmos um índice simples sobre a relação caixa_terminal, utilizando o atributo (cod_cx_term), obtemos um tempo de execução total para a transação Q5 igual a 0.35 s, que representa uma diminuição de 37.5% comparando-se com o tempo na configuração sem o índice.

[5]Oracle Data Loader

Tabela 6.1: Execução do *benchmark SAGRE*

Transação	tempo total	CPU	E/S lógica	E/S física	paginação
Q1	0 00:00:00.56	0:00:00.50	5	0	5
Q2	0 00:00:00.46	0:00:00.42	7	1	5
Q3	0 00:00:00.36	0:00:00.28	5	0	5
Q4	0 00:00:01.05	0:00:00.97	5	1	8
Q5	0 00:00:00.56	0:00:00.44	6	2	5
Q6	0 00:00:00.76	0:00:00.72	6	1	8
Q7	0 00:00:00.39	0:00:00.36	5	1	5
Q8	0 00:00:00.53	0:00:00.38	11	2	5
Q9	0 00:00:00.36	0:00:00.23	6	1	5

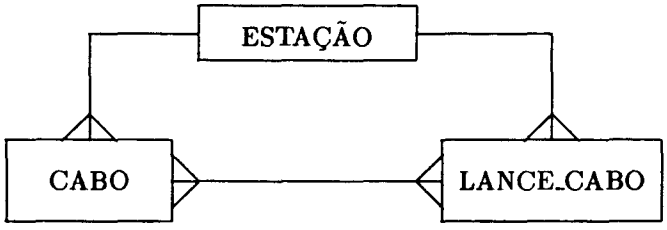

Figura 6.1: Relacionamento entre Estação, Cabo e Lance de Cabo.

Capítulo 7

Conclusão

A *Metodologia de Benchmark Especializado* (MBE) é a principal contribuição deste trabalho. Esta metodologia amplia a abrangência do uso da técnica de *benchmark* para análise de desempenho de sistemas de gerenciamento de bancos de dados em vários sentidos:

- *Estrutura do sistema*: é utilizado um modelo de representação do sistema de banco de dados estruturado em *níveis de abstração*, que pode ser parametrizado pelo usuário, com informações de uma aplicação ou conjunto de aplicações. Nas metodologias tradicionais, que utilizam *benchmark*, a estrutura é especificada diretamente, de maneira artificial.

- *Esquema*: são proporcionadas facilidades de modelagem, onde o usuário pode definir um banco de dados razoavelmente complexo, que se aproxima de situações reais.

- *Conteúdo das bases de dados*: nos métodos tradicionais, o banco de dados utilizado para a análise é construído independentemente das aplicações para as quais se destinam e, os valores utilizados são distribuídos uniformemente, provocando distorções nos resultados obtidos. Na MBE, a geração dos valores leva em consideração as características fornecidas pelos usuários. A caracterização dos atributos, assim como, a caracterização dos domínios são realizadas pelos próprios usuários.

- *Carga de trabalho*: normalmente a carga de trabalho é criada sinteticamente (artificialmente) sem levar em consideração as características específicas do ambiente do usuário. Na MBE pode-se realizar transações com inclusões, atualizações, consultas simples e remoções, e além disso, é possível exercer os seguintes controles:

  1. *Controle de concorrência*: a MBE possui facilidades para configuração do nível de concorrência em sistemas de banco de dados. Esta con-

corrência, definida pelo usuário, pode ser configurada em diferentes níveis de ambiente, tanto externos quanto internos ao SGBD, aumentando-se as possibilidades de adaptação do *benchmark* gerado à realidade.

2. *Controle estruturado de freqüência*: podem ser especificadas freqüências de execução de transações, compondo-se o perfil da carga de trabalho, de acordo com uma estrutura de execução.
3. *Conteúdo das transações*: com a utilização de um paradigma de formulários, permite-se a configuração de transações razoavelmente complexas, aproximando-se de aplicações reais.

A partir da metodologia, foi construída uma ferramenta implementando seus principais componentes. A ferramenta foi validada testando-se duas aplicações, o *benchmark de Wisconsin* e uma aplicação real chamada SAGRE.

A ferramenta, construida, é especializada para suporte de análise de desempenho de sistemas de banco de dados, podendo modelar convenientemente o ambiente operacional do usuário. Ela pode ser utilizada pelo usuário, que desenvolve aplicações e deseja avaliar as alternativas que surgem na fase de *projeto de banco de dados*, assim como, pelo gerente de sistemas, que deseja avaliar o desempenho do SGBD para um conjunto de escolhas de parâmetros de configuração (*system tunning*), ou, ainda, para auxiliar o processo de escolha de um SGBD propriamente.

A execução do *benchmark* transcorreu sem maiores problemas, como era de se esperar, pois a técnica de *benchmark* possibilita a aplicação da ferramenta sobre o sistema (SGBD) real, sem a necessidade de alterações ou de artifícios para sua execução.

A ferramenta desenvolvida é bastante genérica, permitindo a configuração de *benchmarks* de diferentes aplicações. Na ferramenta, pode-se construir um banco de dados razoavelmente complexo, incluindo-se relações com, por exemplo, chaves simples e compostas, chaves estrangeiras, relacionamentos de qualquer cardinalidade e definições de visões dos usuários.

Além de facilidades para a definição de esquemas, a ferramenta possibilita a geração automática de dados, definidos pelo próprio usuário. Os dados obtidos incluem características tais como: configuração de tamanho fixo ou variável, distribuição dos dados uniforme ou não uniforme (distribuição do tipo Zipf), definição de domínios (tipos de dados, intervalo de valores) etc. Tais facilidades diminuem a dificuldade de representação do banco de dados do usuário.

Durante o desenvolvimento deste trabalho, para diminuição da complexidade da implementação do protótipo, não se consideraram alguns aspectos, que podem ser objeto de futuras pesquisas ou de sua continuação:

- *Aplicação a outros modelos*: pode ser interessante desenvolver a metodologia para outros SGBDs que se baseiam em outros modelos que não seja

o modelo relacional. As possíveis extensões podem possibilitar a comparação de diferentes SGBDs, apoiando-se em critérios mais confiáveis, conforme discutido no Capítulo 3.

- *Ampliação da interface*: seria desejável melhorar a interface com o usuário, facilitando a caracterização do *benchmark* para uma determinada aplicação. No interesse de facilitar essa caracterização, poderiam haver configurações *defaults* permitindo-se a obtenção de *benchmarks*, para o caso do usuário não dispor de maiores detalhes sobre sua aplicação, que representassem ambientes típicos de aplicações mais comuns.

- *Ampliação da caracterização de dados*: neste caso, sugere-se a incorporação de outras distribuições de dados, correlacionando-as com tipos de dados ou com o contexto da aplicação do usuário, bem como a possibilidade de se estabelecerem correlações entre atributos e proporcionar novos tipos de dados que ocorram usualmente em aplicações reais (datas, tempo etc.)

- *Ambiente de execução*: o *driver* de execução de transações deve ser estendido para representar um ambiente multi-usuário de fato, onde as várias transações possam ser disparadas por processos independentes, provocando uma concorrência verdadeira no ambiente em que se encontra instalado o SGBD.

- *Bancos de dados distribuídos*: investigar a configuração do ambiente de execução de modo a representar um ambiente de banco de dados distribuídos, onde podem ser especificados o números de servidores, distribuição e particionamento do banco de dados.

- Análise dos resultados: analisar os resultados gerados com a aplicação da MBE, para diferentes SGBDs. Verificando o efeito de distribuições não uniformes, configurações de parâmetros do SGBD, formatos de dados e nível de concorrência presenta na carga de trabalho. Seria interessante fazer testes com SGBDs disponíveis no Brasil, em diferentes plataformas, como por exemplo, computadores de grande porte, minicomputadores e *workstations* de vários fabricantes.

Bibliografia

[ABRA88] Abramowicz K. et al. *Damokles - Reference Manual.* Forschungszentrum Informatik an der Universitat Karlsruhe (Mar. 1988).

[ANDE89] Anderson T.L., Berre A.J. ,Mallison M., Porter H. & Schneider B. The Tektronix HyperModel Benchmark Specification. Tektronix Technical Report No. 89-05, (Aug.1989).

[ANON85] Anon et al. A Measure of the Transaction Processing Power. *Datamation 1*, (Apr. 1985).

[ASTR76] Astrahan M. M. et al. System R: Relational Approach to Database Managment. *ACM Trans. on Database Syst. 1, 2* (June 1976) pp. 97-137.

[BATO82] Batory, D.S. & Gotlieb, C.C. A unifying model of physical databases. *ACM TODS 7, 4* (Dec. 1982) pp. 509-539.

[BENN86] Bennett K. & Cronin D. *SQL*Forms Designers Tutorial, ver. 2.0.* Oracle Corporation, Belmont, CA, USA (1986).

[BITT83] Bitton D., DeWitt D.J., & Turbyfill C. Benchmarking Database Systems, a Systematic Approach. In *Proc. of the VLDB Conference*, Florence, Italy (1983).

[BITT87] Bitton D., Hanraham M.B., & Turbyfill C. Performance of Complex Queries in Main Memory Database Systems. In *Proc. Inter. Conf. on Third Data Engineering*, Los Angeles (Feb. 1987).

[BITT89] Bitton D. & Turbyfill C. A Restrospective on the Wisconsin Benchmark. In Stonebraker M. (ed.), *Readings in Database Systems*, Morgan Kaufmann Pub., San Mateo, CA (1988) pp. 280-299.

[BORA84] Boral H. & DeWitt D.J. A Methodology for Database System Performance Evaluation. In *Proc. of the 1984 SIGMOD Conference*, Boston, MA. (June 1984).

[CHEL90] Chelluri S. & Chadwick D. Benchmarking Committes Grow in Importance. *UnixWorld, Vol.VII, No.5* (Mai. 1990) pp. 119-120.

[CHEN77] Chen P. A Preliminary Framework for Entity-Relationship Models. In *Entity-Relationship Approach to Information Modeling and Analysis*, North-Holland (1983) pp. 19-28.

[CHOU85] Chou T. & Gray J. Transaction Acceleration. it Database Engineering 8, 1, (Mar. 1985) pp.40-52.

[CHRI84] Christodoulakis S. Implications of Certain Assumptions in Database Performance Evaluation. *ACM TODS 9, 2* (jul.1984).

[CODD82] Codd E.F. The 1981 ACM Turing Award Lecture: Relational Database: A Pratical Foundation for Productivity. *Communications of the ACM 25, 2*, (Feb. 1982) pp.109-117.

[COLL87] Colleman D. & Jackson G. DeWitt Benchmark: ORACLE vs. INGRES on Sun. Oracle Corporation Technical Report (Nov. 1987).

[COME79] Comer D. The ubiquitous B-tree. *ACM Comput. Survey 11, 2* (June 1979) pp.121-137.

[DATE86a] Date C.J. *Relational Database: Selected Writings.* Addison-Wesley Pub. Co., Reading, MA. (1986).

[DATE86b] Date C.J. A Critique of the SQL Database Language. In [DATE86a], pp.269-311.

[DATE86c] Date C.J. On The Performance of Relational Database Systems.In [DATE86a], pp.65-76.

[DEMU85] Demurjian S.A. & Hsiao D.K. Benchmarking Database Systems in Multiple Backend Configurations. *Database Engineering 8, 1* (Mar. 1985) pp.29-39.

[DEWI85] DeWitt D.J. Benchmarking Database Systems: Past Effort and Future Directions. *Database Engineering 8, 1* (Mar. 1985) pp.2-9.

[DIMM86] Dimmick S. *ORACLE - Database Administrator's Guide.* Oracle Corporation, Belmont, CA, USA (Apr. 1986).

[DUHL88] Duhl J. & Damon C. A Performance Comparison of Object and Relational Databases Using the Sun Benchmark. In *OOPSLA 1988 Proceedings*, ACM (1988).

[EFFE84] Effelsberg W. & Haerder T. Principles of Database Buffer Management. *ACM TODS 9, 4* (Dec. 1984).

[FINK88] Finkelstein S, Schkolnick M. & Tibério P. Physical Database Design for Relational Databases. *ACM TODS 13, 1* (Mar. 1988) pp.91-128.

[FINK90] Finkelstein R. Benchmark Wars - What TP1 benchmark tests can and can't tell you about a DBMS. *DBMS 3, 2* (Feb. 1990) pp.14-15.

[GANE79] Gane C. & Sarson T. Análise Estruturada de Sistemas. Editora Livros Técnicos e Científicos, 1a. ed. (1983).

[GOTL80] Gotlieb C.C. Some Large Questions about Very Large Data Bases. In *Proc. of 6th. International Conference on VLDB* (Oct. 1980).

[HAWT85] Hawthorn P. Variations on a Benchmark. *Database Engineering 8, 1*, (Mar. 1985) pp.19-28.

[KNUT73] Knuth D.E. *The Art of Computer Programming, vol.1: Fundamental Algorithms.* Addison-Wesley Pub. Co. Readings, MA. (1973) 722 pp.

[KORT89] Korth H.F. *Sistemas de Bancos de Dados.* McGraw-Hill, São Paulo (1989) 582 pp.

[LOCH78] Lochovsky F.H. Data Base Management System User Performance. Ph.D. Thesis, Dept. of Computer Sciences, Univ. of Toronto (1978).

[LYNC88] Lynch C.A. Selectivity Estimation and Query Optimization in Large Databases with Highly Skewed Distributions of Columns Values. In *Proc. of the 14th VLDB Conference*, Los Angeles (1988) pp.240-251.

[MAGA81] Magalhães G.C. Improving the Performance of Database Systems. Ph.D. Thesis, Dept. of Computer Sciences, Univ. of Toronto (1981).

[MANN88] Mannino M.V., Chu P. & Sager T. Statistical Profile Estimation in Database Systems. *ACM Comput. Surveys 20, 3* (Sept. 1988) pp.191-221.

[MCLA88] McLaughlin M.E. et al. An Integrated Methodology and Toolset for Database Design. *SIGMOD Record 17, 4* (Dec. 1988) pp.37-55.

[MELO88] Melo C.H. Uma ferramenta de auxílio ao projeto físico de bancos de dados relacionais. Dissertação de Mestrado, Dept. de Ciência da Computação, ICEX-UFMG (1988).

[MELO89] Melo C.H. & Laender A.H.F. Um Modelo para Projeto Fisico de Bancos de Dados Relacionais. *Anais do 4o. Simpósio Brasileiro de Banco de Dados*, Campinas (1989) pp.197-208.

[MOHA89] Mohan C., Haderle D., Wang Y. & Cheng J. Single Table Access Using Multiple Indexes: Optimization, Execution, and Concurrency Control Techniques. IBM Research Report, Almaden Research Center, Data Base Tech. Institute, San Jose, CA, USA, e-mail *mohan@ibm.com.*

[NOGU89] Nogueira M.L. Um Sistema para Manuseio de Objetos Entidade-Relacionamento no Modelo Relacional. Dissertação de Mestrado, Dept. Ciência da Computação, Unicamp, Campinas, SP (Mar. 1989).

[ORAC86] *SQL*Plus User's Guide.* Oracle Corporation, Belmont, CA, USA (1986).

[PASC87] *Turbo Pascal - Owner's Handbook, ver. 4.0.* Borland International Inc. (1987).

[PERE89] Pereira R.F. & Magalhães G.C. Análise de Desempenho de Bancos de Dados utilizando uma Metodologia de Benchmark Especializado. *Anais do 4o. Simpósio Brasileiro de Banco de Dados*, Campinas, SP, (abril 1989) pp.292-293.

[RADE89] RADE: Relatório para Avaliação do Desempenho Empresarial. Relatório técnico, Telebrás (Jul. 1989).

[RAPO81] Raposo I.C. Analytic Modeling of Data Base Systems: The Design of a System 2000 Performance Predictor. Ms.C. Thesis, Dept. of Computer Sciences, Univ. of Toronto (1981).

[RAPO86] Raposo I.C. Prophet: A Layered Analytical Model for Performance Prediction of Database Systems. Ph.D. Thesis, Dept. of Computer Sciences, Univ. of Toronto (1986).

[RUBE87] Rubenstein W.B., Kubicar M.S. & Catell R.G.G. BenchMarking Simple Database Operations. In *SIGMOD Proceedings (1987). SIGMOD Record 16, 3* (Dec. 1987).

[SAGR90] Sistema Automatizado de Gerenciamento de Rede Externa SAGRE. Relatório Técnico No. 205-001-700, Telebrás (Jan. 1990).

[SAKT76] Sakti P.G. & Tuel W.G. A Design an Experiment to Model Data Base System Performance. *IEEE Transactions on Software Engineering* (June 1976) pp.97-106.

[SALZ89] Salza S. & Terranova M. Evaluating the Size of Queries on Relational Databases with Non Uniform Distribution and Stochastic Dependence. *ACM TODS* (1989).

[SELI79] Selinger P.G., Astrahan M.M., Chamberlin D.D., Lorie R.A. & Price T.G. Access Path Selection in a Relational Database Management System. In *Proc. ACM-SIGMOD International Conference on Management of Data* (Boston, June 1979).

[SEVC81] Sevcik K.C. Database System Performance Prediction Using an Analytical Model. In *Proc. 7th. VLDB International Conference* (Cannes, Sept. 1981).

[SILV90] Silva E.C., Laender A.H.F. & Braga J.L. Otimização de consultas a bancos de dados relacionais dirigida por base de conhecimento multinível. *Revista Brasileira de Computação 5, 4*, (Abr./Jun. 1990) pp.37-44.

[STAN87] Stanley Y.W., Jozo D., Batory D.S., Navathe S.B. & Elnicki R. A Cost-Benefit Decision Model: Analysis, Comparison, and Selection of Data Management Systems. *ACM TODS 12, 3* (Sept. 1987) pp.472-520.

[STON76] Stonebraker M. The Design and Implementation of INGRES. *ACM TODS 1, 3* (Sept. 1976) pp.189-222.

[STON82] Stonebraker M. et. al. Performance Analysis of Distributed Database Systems. Database Engineering.

[STON85] Stonebraker M. Tips on Benchmarking DataBase Systems. *DataBase Engineering 4, 1* (Mar. 1985) pp.10-18.

[STON86] Stonebraker M. & Rowe, L. The design of Postgres. *ACM-SIGMOD Proceedings of the International Conference on the Management of Data*, Washington, D.C. (May 1986).

[TAND88] Tandem Database Group. A Benchmark of NonStop SQL on the Debit Credit Transaction. Tandem Tech. Report 87.4, Tandem Computer Inc. (1988).

[TOOL85] *Turbo Database Toolbox - Owner's Handbook.* Borland International Inc. (1985).

[TPC89] TPC Benchmark A - Draft 6-PR Proposed Standard. Transaction Processing Council - TPC, Los Altos, CA. (Aug. 1989).

[TURB87] Turbyfil C. Comparative Benchmarking of Relational Database Systems. Ph.D. Thesis, Dept. of Computer Sciences, Cornell Univ., Ithaca, NY. (1987).

[ULLM82] Ullman J.D. *Principles of Database Systems, 2nd. ed.* Computer Science Press,Rockville, MD. (1982) 494 pp.

[WART86] Wartik P.S. & Penedo M. Fillin: A Reusable Tools for Form-Oriented Software. IEEE Software (Mar. 1986) pp.61-69.

[WIED83] Wiederhold G. *Database Design, 2nd. ed.* McGraw-Hill (1983) 751 pp.

[WILM89] Wilmot R.B. File Usage Patterns from SMF Data: Highly Skewed Usage. Amdahl Co. Tech. Report, (1989).

[YU89] Yu P.S. et al. Workload Characterization of Relational Environments. IBM Tech. Report RC 14675(#65770), T.J. Watson Research Center (June 1989).

[ZDON90] Zdonik S. & Maier D. Object-Oriented Fundamentals. In *Readings in Object-Oriented Database Systems*, Morgan Kaufmann Pub. Inc., Palo Alto, CA, USA (1990).

Apêendice A

Benchmark de Wisconsin na MBE

Configuração do Benchmark de Wisconsin adaptado à MBE.

Relações que compõem o Benchmark de Wisconsin: onek, tenk1 e tenk2

Visões associadas:

view_tenk1	select from tenk1
JoinAselB	select * from tenk1, tenk2 where (tenk1.unique2 = tenk2.unique2) and (tenk2.unique2 < 1000)
Bprime	select * from tenk1 where tenk1.unique2 < 1000
JoinABprime	select * from tenk1, Bprime where tenk1.unique2 = Bprime.unique2
JoinCselAselB	select * from onek, tenk1 where (onek.unique2 = tenk1.unique2) and (tenk1.unique2 = tenk2.unique2) and (tenk1.unique2 < 1000) and (tenk2.unique2 < 1000)
SJoinAselB	select from tenk1, tenk2 where (tenk1.unique1 = tenk2.unique1) and (tenk2.unique1 < 1000)
sJoinABprime	select from tenk1, Bprime where tenk1.unique1 = Bprime.unique1
sJoinCselAselB	select from onek, tenk1 where (onek.unique1 = tenk1.unique1) and (tenk1.unique1 = tenk2.unique1) and (tenk1.unique1 < 1000) and (tenk2.unique1 < 1000)

Apêendice B

Estrutura do dicionário

O dicionário de dados foi implementado na ferramenta como arquivos no sistema operacional. São as seguintes, as definições em Pascal desses arquivos:

```
const

  VMAX      = 16000;{ 64K  tamanho maximo do bitmap       }
  BLOCKSIZE = 16000;{ 2 * VMAX    tamanho maximo de bloco }
  NUMBLOCKS = 1;    { Numero de blocos de memoria         }
  MAXKEYNUM = 5;    { Numero atribs max.  chave  composta }
  NULL      = '';

{ definicao de tipos para o benchmark }
type

  StrName  = string[20];
  Str80    = string[80];
  Str255   = string;
  TransCode= string[26];

  DataTyp         = ( charsy, numbersy, datesy );
  OrderTyp        = ( randomsy, sequential, cicle, rotating );
  DistributionTyp = ( uniform, zipf, user, aleatorio );
  GetValueTyp     = ( auto, modulo, even, odd, lista );
  QueryTyp        = ( consulta, inclusao );
  AgeTyp          = ( newsy, oldsy );

  KeyEntry = record
    atribute    : strname;
```

```
    rel_subkey  : strname;
    KeyId       : integer;
    value       : string
  end;

  KeyArrayTyp = array[1..MAXKEYNUM] of KeyEntry;

  BitMap       = array[0..BLOCKSIZE-1] of boolean;
  BitMapPtrTyp = ^BitMap;
  MemoBlock    = array[0..BLOCKSIZE-1] of longint;
  MemoBlkPtr   = ^MemoBlock;
  ArrBlkPtrTyp = array[1..4] of MemoBlkPtr;

{--------- MODELO DE REPRESENTACAO DOS DADOS ----------}


RelationRecTy = record
  status       : integer;
  nome_rel     : StrName;
  cardinalidade: longint;
  num_atribs   : integer;
  nome_cluster : StrName
end;

ClusterRecTy = record
  status       : integer;
  nome_cluster : StrName;
  nome_atrib   : StrName;
  tipo_dado    : StrName
end;

AtributeRecTy = record
  status          : integer;
  nome_rel        : StrName;
  nome_atrib      : StrName;
  nome_dominio    : StrName;
  nulo            : integer;
  distribuicao    : StrName;
  unico           : string[6];
  ordem_cluster   : integer;
  disp_dados      : StrName;
  n_distintos     : string[9];
  restricao       : integer;
  min             : StrName;
```

```
  max             : StrName;
  tamanho_min     : integer;
  tamanho_max     : integer;
  obtem_valor     : StrName;
  prox_pos        : longint;
  ja_criado       : boolean
end;

DomainRecTy = record
  status          : integer;
  nome_dom        : StrName;
  tipo_dado       : StrName;
  formato         : StrName;
  n_distintos     : string[6];
  min             : string[9];
  max             : string[9];
  tamanho_min     : integer;
  tamanho_max     : integer;
  obtem_valor     : StrName;
  parametro       : integer;
end;

KeyRecTy = record
  status     : integer;
  nome_rel   : strName;
  id_subkey  : integer;
  nome_atrib: strName;
  rel_subkey: strname;
end;

ForeignKeyRecTy = record
  status          : integer;
  nome_rel_est    : StrName;
  nome_atrib_est  : StrName;
  nome_rel_prim   : StrName;
  id_chave_prim   : integer;
  nome_atrib_prim: StrName
end;

ListRecTy = record
  status      : integer;
  nome_dom    : StrName;
  criador     : StrName;
  num_item    : integer;
```

```
  frequencia : integer
end;

FrequencyRecTy = record
  status     : integer;
  nome_rel   : StrName;
  nome_atrib : StrName;
  num_item   : integer;
  valor      : StrName
end;

IndexRecTy = record
  status      : integer;
  nome_indx   : StrName;
  nome_rel    : StrName;
  nome_atribs : StrName;
  index_type  : StrName;
  iord        : StrName;
  num_seq     : integer
end;

{--------- MODELO DE TRANSACAO ---------}

AplicationRecTy = record
  status      : integer;
  nome_aplic  : StrName;
  descricao   : string[200];
  freq_aplic  : integer
end;

BaseViewRecTy   = record
  status      : integer;
  nome_visao  : StrName;
  codigo_SQL  : string;
end;

WindowRecTy      = record
  status             : integer;
  nome_aplic         : StrName;
  num_janela         : integer;
  nome_visao         : StrName;
  num_trans_section  : integer;
  num_sec_aplic      : integer;
  freq_sel           : integer;
```

```
  where_set_up      : String ;   { cond1^cond2^...^condn^ }
end;

FieldsRecTy   = record
  status        : integer;
  nome_aplic    : StrName;
  num_janela    : integer;
  num_campo     : integer;
  nome_rel_base: StrName;
  nome_atrib    : StrName;
end;

SelectRecTy   = record
  status          : integer;
  nome_aplic      : StrName;
  num_janela      : integer;
  num_sel         : integer;
  freq_parc       : integer;
  perc_upt        : integer;
  perc_del        : integer;
  distinct_expr   : string;
  group_by_expr   : string;
  order_by_expr   : string;
  where_tipo      : string;   { cond1^cond2^...^condn^ }
end;

InsertRecTy   = record
  status       : integer;
  nome_aplic   : StrName;
  num_janela   : integer;
  num_ins      : integer;
  freq_parc    : integer;
  cmd_ins      : string
end;

UpdateRecTy   = record
  status       : integer;
  nome_aplic   : StrName;
  num_janela   : integer;
  num_sel      : integer;
  campos       : string   { cond1^cond2^...^condn^} { campos para update}
end;
```

Apêendice C

Benchmark **SAGRE**

```
/*  ======= == ============ == ==== == =====  */
/*  Arquivo de Configuracao da Base de Dados  */
/*  ======= == ============ == ==== == =====  */

/*

   Prefixos das linhas:

   Os dados sao formatados em cada linha para entrada
   nas tabelas do dicionario de  dados   conforme   o
   prefixo da linha atraves de um utilitario especifico.

   RELAC  relacao
   ATRIB  atributo
   CHAVE  chave
   FRKEY  chave estrangeira
   DOMIN  dominio

*/

/* --- Estacao e Secao de Servico --- */

RELACestacao#2#6#
RELACsecao_servico#5#3#

ATRIBestacao#cod_est#numero##UNIFORME#UNICO##RANDOM#10#
>>#0#9###AUTO
ATRIBestacao#nome_est#string##UNIFORME#UNICO##
```

```
>>SEQUENTIAL#10####25#25#AUTO
ATRIBestacao#tipo_est#estacao##UNIFORME###RANDOM#
ATRIBestacao#capac_final#numero##ALEATORIO###RANDOM#10#
>>#40000#50000###AUTO
ATRIBestacao#capac_atual#numero##ALEATORIO###RANDOM#10#
>>#20000#40000###AUTO
ATRIBestacao#cod_logr#endereco##UNIFORME#UNICO##RANDOM#
>>10######AUTO

ATRIBsecao_servico#cod_ss#numero##UNIFORME###RANDOM#15#
>>#0#14###AUTO
ATRIBsecao_servico#cod_est###UNIFORME###RANDOM#######
ATRIBsecao_servico#area#numero##ALEATORIO###RANDOM#1000
>>0##3000#100000###AUTO

CHAVEestacao#1#cod_est#estacao

CHAVEsecao_servico#1#cod_ss#secao_servico
CHAVEsecao_servico#2#cod_est#estacao

FRKEYsecao_servico#cod_est#estacao#1#cod_est

RELACarmario#5#8

ATRIBarmario#cod_ss#
ATRIBarmario#cod_est#
ATRIBarmario#endereco#endereco##UNIFORME###RANDOM#
ATRIBarmario#colocacao#loc_arm##UNIFORME###RANDOM#
ATRIBarmario#tipo_armario#armario##UNIFORME###RANDOM#
ATRIBarmario#tipo_bloco#bloco##UNIFORME###RANDOM#
ATRIBarmario#capac_alim#numero##ALEATORIO###RANDOM#50#
>>#0#49###AUTO
ATRIBarmario#capac_distr#numero##ALEATORIO###RANDOM#50#
>>#0#49
###AUTO

CHAVEarmario#1#cod_ss#secao_servico
CHAVEarmario#1#cod_est#secao_servico

FRKEYarmario#cod_est#estacao#1#cod_est
FRKEYarmario#cod_ss#secao_servico#1#cod_ss

RELACcaixa_subterranea#30#4
RELACcaixa_terminal#20#5
```

```
ATRIBcaixa_subterranea#cod_cx_subt#numero##UNIFORME###
>>RANDOM#100##0#99###AUTO
ATRIBcaixa_subterranea#cod_est###UNIFORME###RANDOM#
ATRIBcaixa_subterranea#tipo_caixa#cx_subt##UNIFORME###
>>RANDOM#
ATRIBcaixa_subterranea#endereco#endereco##UNIFORME#UNI
>>CO##RANDOM

ATRIBcaixa_terminal#cod_cx_term#numero##UNIFORME###RAN
>>DOM#200##0#199###AUTO
ATRIBcaixa_terminal#cod_ss###UNIFORME###RANDOM#
ATRIBcaixa_terminal#cod_est###UNIFORME###RANDOM#
ATRIBcaixa_terminal#tipo_bloco#bloco##UNIFORME###RANDOM
ATRIBcaixa_terminal#endereco#endereco##UNIFORME#UNICO##
>>RANDOM

CHAVEcaixa_subterranea#1#cod_cx_subt#caixa_subterranea
CHAVEcaixa_subterranea#2#cod_est#estacao

CHAVEcaixa_terminal#1#cod_cx_term#caixa_terminal
CHAVEcaixa_terminal#2#cod_ss#secao_servico
CHAVEcaixa_terminal#2#cod_est#secao_servico

FRKEYcaixa_subterranea#cod_est#estacao#1#cod_est

FRKEYcaixa_terminal#cod_est#estacao#1#cod_est
FRKEYcaixa_terminal#cod_ss#secao_servico#1#cod_ss

RELACvertical#10#5
RELACcabo#20#6
RELAClance_cabo#50#8
RELACdg#3#7
RELACdg_predial#20#7

ATRIBvertical#cod_vert#numero##UNIFORME###RANDOM#5##0#4
>>###AUTO
ATRIBvertical#cod_dg###UNIFORME###RANDOM#
ATRIBvertical#cod_est###UNIFORME###RANDOM#
ATRIBvertical#tipo_bloco#bloco##UNIFORME###RANDOM#
ATRIBvertical#qtde_bloco#numero##ALEATORIO###RANDOM#100
>>00##5#200###AUTO

CHAVEvertical#1#cod_vert#vertical
```

```
CHAVEvertical#2#cod_dg#dg
CHAVEvertical#2#cod_est#dg

FRKEYvertical#cod_est#estacao#1#cod_est
FRKEYvertical#cod_dg#dg#1#cod_dg

ATRIBcabo#cod_cabo#numero##UNIFORME###RANDOM#100##0#99#
>>##AUTO
ATRIBcabo#cod_vert###UNIFORME###RANDOM#
ATRIBcabo#cod_dg###UNIFORME###RANDOM#
ATRIBcabo#cod_est###UNIFORME###RANDOM#
ATRIBcabo#nat_cabo#natureza_cabo##UNIFORME###RANDOM
ATRIBcabo#capac_cabo#numero##ALEATORIO###RANDOM#1900##1
>>00#2000###AUTO

CHAVEcabo#1#cod_cabo#cabo
CHAVEcabo#2#cod_vert#vertical
CHAVEcabo#2#cod_dg#vertical
CHAVEcabo#2#cod_est#vertical

FRKEYcabo#cod_est#estacao#1#cod_est
FRKEYcabo#cod_dg#dg#1#cod_dg
FRKEYcabo#cod_vert#vertical#1#cod_vert

ATRIBlance_cabo#cod_lance_cabo#numero##UNIFORME###RANDO
>>M#200##0#199###AUTO
ATRIBlance_cabo#compr_lance#numero##ALEATORIO###RANDOM#
>>10000##100#2000###AUTO
ATRIBlance_cabo#tipo_cabo#cabo##UNIFORME###RANDOM#
ATRIBlance_cabo#diam_condutor#numero##ALEATORIO###RANDO
>>M#10000##10#100###AUTO
ATRIBlance_cabo#capac_cabo#numero##ALEATORIO###RANDOM#1
>>00##50#1800###AUTO
ATRIBlance_cabo#cont_inicial#numero##ALEATORIO###RANDOM
>>#1000##100#1000###AUTO
ATRIBlance_cabo#cont_final#numero##ALEATORIO###RANDOM#1
>>000##1001#1900###AUTO
ATRIBlance_cabo#pares_mortos#numero##ALEATORIO###RANDOM
>>#200##1#190###AUTO

CHAVElance_cabo#1#cod_lance_cabo#lance_cabo

FRKEYlance_cabo#cod_est#estacao#1#cod_est
```

```
ATRIBdg#cod_dg#numero##UNIFORME###RANDOM#10##0#9###AUTO
ATRIBdg#cod_est###UNIFORME###RANDOM#
ATRIBdg#tipo_dg#dg##UNIFORME###RANDOM#
ATRIBdg#total_verticais#numero##ALEATORIO###RANDOM#10##0
>>#9###AUTO
ATRIBdg#verticais_instaladas#numero##ALEATORIO###RANDOM#
>>10##0#9###AUTO
ATRIBdg#cap_projeto#numero##ALEATORIO###RANDOM#10##0#9##
>>#AUTO
ATRIBdg#tipo_bloco_proj#bloco##UNIFORME###RANDOM#

CHAVEdg#1#cod_dg#dg
CHAVEdg#2#cod_est#estacao

FRKEYdg#cod_est#estacao#1#cod_est

ATRIBdg_predial#cod_dg_predial#numero##UNIFORME###RANDOM#
>>20##0#19###AUTO
ATRIBdg_predial#cod_ss###UNIFORME###RANDOM#
ATRIBdg_predial#cod_est###UNIFORME###RANDOM#
ATRIBdg_predial#tipo_dg_predial#dg_predial##UNIFORME###RA
>>NDOM#
ATRIBdg_predial#tipo_bloco#bloco##UNIFORME###RANDOM#
ATRIBdg_predial#qtde_bloco#numero##ALEATORIO###RANDOM#10#
>>#0#9###AUTO
ATRIBdg_predial#endereco#endereco##UNIFORME###RANDOM#

CHAVEdg_predial#1#cod_dg_predial#dg_predial
CHAVEdg_predial#2#cod_ss#secao_servico
CHAVEdg_predial#2#cod_est#secao_servico

FRKEYdg_predial#cod_est#estacao#1#cod_est
FRKEYdg_predial#cod_ss#secao_servico#1#cod_ss

RELACduto#100#4
RELACemenda#50#3
RELACposte#20#2
RELAClance_duto#50#6

/* ---- Relacionamentos --- */

RELACemenda_poste#20#2
RELACemenda_caixasub#30#3
```

```
ATRIBlance_duto#cod_lance_duto#numero##UNIFORME###RANDO
>>M#150##0#149###AUTO
ATRIBlance_duto#compr_pp#numero##ALEATORIO###RANDOM#100
>>00##100#2000###AUTO
ATRIBlance_duto#compr_cc#numero##ALEATORIO###RANDOM#100
>>00##100#2000###AUTO
ATRIBlance_duto#quant_duto#numero##ALEATORIO###RANDOM#2
>>4##6#24###AUTO
ATRIBlance_duto#diam_duto#numero##ALEATORIO###RANDOM#10
>>000##10#30###AUTO
ATRIBlance_duto#tipo_duto#duto##UNIFORME###RANDOM#

CHAVElance_duto#1#cod_lance_duto#lance_duto

ATRIBduto#cod_duto#numero##UNIFORME###RANDOM#6##0#5###A
>>UTO
ATRIBduto#cod_lance_duto###UNIFORME###RANDOM#
ATRIBduto#status#ocupacao##UNIFORME###RANDOM#
ATRIBduto#cod_lance_cabo###UNIFORME###RANDOM#

CHAVEduto#1#cod_duto#duto
CHAVEduto#2#cod_lance_duto#lance_duto

FRKEYduto#cod_lance_duto#lance_duto#1#cod_lance_duto
FRKEYduto#cod_lance_cabo#lance_cabo#1#cod_lance_cabo

ATRIBemenda#cod_emenda#numero##UNIFORME###RANDOM#200##0
>>#199###AUTO
ATRIBemenda#tipo_emenda#emenda##UNIFORME###RANDOM#
ATRIBemenda#tipo_conector#conector##ZIPF###RANDOM#

CHAVEemenda#1#cod_emenda#emenda

ATRIBposte#cod_poste#numero##UNIFORME###RANDOM#300##0#2
>>99###AUTO
ATRIBposte#tipo_poste#poste##UNIFORME###RANDOM#

CHAVEposte#1#cod_poste#poste

ATRIBemenda_poste#cod_emenda###UNIFORME###RANDOM#
ATRIBemenda_poste#cod_poste###UNIFORME###RANDOM#

CHAVEemenda_poste#1#cod_emenda#emenda
CHAVEemenda_poste#2#cod_poste#poste
```

```
FRKEYemenda_poste#cod_emenda#emenda#1#cod_emenda
FRKEYemenda_poste#cod_poste#poste#1#cod_poste

ATRIBemenda_caixasub#cod_emenda###UNIFORME###RANDOM#
ATRIBemenda_caixasub#cod_cx_subt###UNIFORME###RANDOM#
ATRIBemenda_caixasub#cod_est###UNIFORME###RANDOM#

CHAVEemenda_caixasub#1#cod_emenda#emenda
CHAVEemenda_caixasub#2#cod_cx_subt#caixa_subterranea
CHAVEemenda_caixasub#2#cod_est#caixa_subterranea

FRKEYemenda_caixasub#cod_emenda#emenda#1#cod_emenda
FRKEYemenda_caixasub#cod_cx_subt#caixa_subterranea#1#co
>>d_cx_subt
FRKEYemenda_caixasub#cod_est#estacao#1#cod_est

/* --- Relacionamentos Lance Cabo --- */

RELAClancecabo_cabo#100#5#
RELAClancecabo_armario#10#3
RELAClancecabo_cxterm#20#4
RELAClancecabo_dgpredial#20#4

ATRIBlancecabo_cabo#cod_lance_cabo###UNIFORME###RANDOM#
ATRIBlancecabo_cabo#cod_cabo###UNIFORME###RANDOM#
ATRIBlancecabo_cabo#cod_vert###UNIFORME###RANDOM#
ATRIBlancecabo_cabo#cod_dg###UNIFORME###RANDOM#
ATRIBlancecabo_cabo#cod_est###UNIFORME###RANDOM#

CHAVElancecabo_cabo#1#cod_lance_cabo#lance_cabo
CHAVElancecabo_cabo#2#cod_cabo#cabo
CHAVElancecabo_cabo#2#cod_vert#cabo
CHAVElancecabo_cabo#2#cod_dg#cabo
CHAVElancecabo_cabo#2#cod_est#cabo

FRKEYlancecabo_cabo#cod_lance_cabo#lance_cabo#1#cod_lan
>>ce_cabo
FRKEYlancecabo_cabo#cod_est#estacao#1#cod_est
FRKEYlancecabo_cabo#cod_dg#dg#1#cod_dg
FRKEYlancecabo_cabo#cod_vert#vertical#1#cod_vert
FRKEYlancecabo_cabo#cod_cabo#cabo#1#cod_cabo

ATRIBlancecabo_armario#cod_lance_cabo###UNIFORME###RAND
```

```
>>OM#
ATRIBlancecabo_armario#cod_ss###UNIFORME###RANDOM#
ATRIBlancecabo_armario#cod_est###UNIFORME###RANDOM#

CHAVElancecabo_armario#1#cod_lance_cabo#lance_cabo
CHAVElancecabo_armario#2#cod_ss#armario
CHAVElancecabo_armario#2#cod_est#armario

FRKEYlancecabo_armario#cod_lance_cabo#lance_cabo#1#cod_
>>lance_cabo
FRKEYlancecabo_armario#cod_ss#secao_servico#1#cod_ss
FRKEYlancecabo_armario#cod_est#estacao#1#cod_est

ATRIBlancecabo_cxterm#cod_lance_cabo###UNIFORME###RANDO
>>M#
ATRIBlancecabo_cxterm#cod_cx_term###UNIFORME###RANDOM#
ATRIBlancecabo_cxterm#cod_ss###UNIFORME###RANDOM#
ATRIBlancecabo_cxterm#cod_est###UNIFORME###RANDOM#

CHAVElancecabo_cxterm#1#cod_lance_cabo#lance_cabo
CHAVElancecabo_cxterm#2#cod_cx_term#caixa_terminal
CHAVElancecabo_cxterm#2#cod_ss#caixa_terminal
CHAVElancecabo_cxterm#2#cod_est#caixa_terminal

FRKEYlancecabo_cxterm#cod_lance_cabo#lance_cabo#1#cod_l
>>ance_cabo
FRKEYlancecabo_cxterm#cod_cx_term#caixa_terminal#1#cod_
>>cx_term
FRKEYlancecabo_cxterm#cod_ss#secao_servico#1#cod_ss
FRKEYlancecabo_cxterm#cod_est#estacao#1#cod_est

ATRIBlancecabo_dgpredial#cod_lance_cabo###UNIFORME###RA
>>NDOM#
ATRIBlancecabo_dgpredial#cod_dg_predial###UNIFORME###RA
>>NDOM#
ATRIBlancecabo_dgpredial#cod_ss###UNIFORME###RANDOM#
ATRIBlancecabo_dgpredial#cod_est###UNIFORME###RANDOM#

CHAVElancecabo_dgpredial#1#cod_lance_cabo#lance_cabo
CHAVElancecabo_dgpredial#2#cod_dg_predial#dg_predial
CHAVElancecabo_dgpredial#2#cod_ss#dg_predial
CHAVElancecabo_dgpredial#2#cod_est#dg_predial

FRKEYlancecabo_dgpredial#cod_lance_cabo#lance_cabo#1#co
```

```
>>d_lance_cabo
FRKEYlancecabo_dgpredial#cod_dg_predial#dg_predial#1#co
>>d_dg_predial
FRKEYlancecabo_dgpredial#cod_ss#secao_servico#1#cod_ss
FRKEYlancecabo_dgpredial#cod_est#estacao#1#cod_est

/* --- DOMINIOS --- */

DOMINnumero#NUMBER
DOMINstring#CHAR
DOMINbloco#CHAR#FIXO#40###5#5#AUTO
DOMINposte#CHAR#FIXO#30###5#5#AUTO
DOMINcabo#CHAR#FIXO#25###5#5#AUTO
DOMINdg_predial#CHAR#FIXO#10###5#5#AUTO
DOMINduto#CHAR#FIXO#10###5#5#AUTO
DOMINsubduto#CHAR#FIXO#10###5#5#AUTO
DOMINocupacao#CHAR#VARIAVEL#3###4#6#AUTO
DOMINarmario#CHAR#FIXO#20###5#5#AUTO
DOMINloc_arm#CHAR#VARIAVEL#2###5#8#AUTO
DOMINnatureza_cabo#CHAR#FIXO#3###4#4#AUTO
DOMINcaixa_term#CHAR#VARIAVEL#20###5#10#AUTO
DOMINemenda#CHAR#FIXO#10###4#4#AUTO
DOMINconector#CHAR#VARIAVEL#15###5#10#AUTO
DOMINcx_subt#CHAR#VARIAVEL#10###5#10#AUTO
DOMINestacao#CHAR#FIXO#20###5#5#AUTO
DOMINendereco#CHAR#VARIAVEL#10000###15#30#AUTO
DOMINdg#CHAR#FIXO#10###5#5#AUTO
DOMINorigem_destino#FIXO#2###4#4#AUTO

/*   ======= == ============ == ===== == ======== */
/*   Arquivo de configuracao da carga de trabalho */
/*   ======= == ============ == ===== == ======== */

/*

   Prefixos das linha:

   Os dados sao formatados em cada linha para entrada
   nas tabelas do dicionario de dados conforme o prefixo
   da linha.

   APLIC  aplicacao
   VISAO  visao base
   JANEL  janela
```

```
   CAMPO  campo da janela
   SELEC  selecao


*/

APLICsagre#Sistema de Gerenciamento de Rede Externa#1

/* Q1: Quais os cabos que chegam em uma vertical? */

VISAOvisao1#
select V.cod_est,
       V.cod_dg,
       V.cod_vert,
       nat_cabo, cod_cabo, capac_cabo
from   cabo C, vertical V
where  V.cod_est  = C.cod_est
  and  V.cod_dg   = C.cod_dg
  and  V.cod_vert = C.cod_vert #

JANELsagre#1#visao1#1#10#100#

CAMPOsagre#1#1#cabo#cod_est#
CAMPOsagre#1#2#cabo#cod_vert

SELECsagre#1#1#100#0#0####2^

/* Q2: Qual e' a contagem de um lance de cabo? */

VISAOLanceCabo#
select cod_lance_cabo,
       cont_inicial,
       cont_final,
       pares_mortos
from   lance_cabo #

JANELsagre#3#LanceCabo#1#10#100#
CAMPOsagre#3#1#lance_cabo#cod_lance_cabo#
SELECsagre#3#1#100#0#0####1^

/* Q3: Quais cabos estao em um lance de cabo ? */

VISAOJoinLanceCabo#
select L.cod_lance_cabo,
```

```
       cod_est,
       cod_dg,
       cod_vert,
       cod_cabo
from   lance_cabo L,
       cabo C,
       lancecabo_cabo R
where  L.cod_lance_cabo=R.cod_lance_cabo
  and  R.cod_est  = C.cod_est
  and  R.cod_dg   = C.cod_dg
  and  R.cod_vert = C.cod_vert
  and  R.cod_cabo = C.cod_cabo #

JANELsagre#4#JoinLanceCabo#1#10#100#
CAMPOsagre#4#1#lancecabo_cabo#cod_lance_cabo#
SELECsagre#4#1#100#0#0###ORDER BY C.cod_est, cod_dg, cod_
>>vert, cod_cabo#1^

/* Q4: Qual a localizacao de uma caixa terminal ? */

VISAOCxTerm#
select * from caixa_terminal #

JANELsagre#5#CxTerm#1#10#100#1^

CAMPOsagre#5#1#caixa_terminal#cod_ss#
CAMPOsagre#5#2#caixa_terminal#cod_cx_term#
SELECsagre#5#1#100#0#0###ORDER BY cod_cx_term#2^

/* Q5: Qual a contagem de um dg predial ? */

VISAOJoinLanceDGpredial#
select L.cod_lance_cabo,
       D.cod_dg_predial,
       R.cod_ss,
       R.cod_est,
       cont_inicial,
       cont_final
from   Lance_Cabo L, LanceCabo_DgPredial R, Dg_Predial D
where  L.cod_lance_cabo = R.cod_lance_cabo
  and  R.cod_dg_predial = D.cod_dg_predial
  and  R.cod_ss = D.cod_ss
  and  R.cod_est = D.cod_est #
```

```
JANELsagre#6#JoinLanceDGpredial#1#10#100#
CAMPOsagre#6#1#lance_cabo#cod_lance_cabo#
SELECsagre#6#1#100#0#0####1^

/* Q6: Que caixa terminal atende um determinado
       endereco ? */

JANELsagre#7#CxTerm#1#10#100#
CAMPOsagre#7#1#caixa_terminal#endereco#
SELECsagre#7#1#100#0#0####1^

/* Q7: Pesquise os dados de uma estacao? */

VISAOEstacao_view#
select * from estacao #

JANELsagre#8#Estacao_view#1#10#100
CAMPOsagre#8#1#estacao#cod_est#
SELECsagre#8#1#100#0#0####1^

/* Q8: Pesquise os dados de secoes de servico
       para uma estacao dada. */

VISAOSS_view#select * from secao_servico #

JANELsagre#9#SS_view#1#10#100
CAMPOsagre#9#1#estacao#cod_est#
SELECsagre#9#1#100#0#0####1^

/* Q9: Qual a quantidade de pares terminados em uma
       estacao? */

VISAOJoinEstCabo#
select E.cod_est,
       nome_est,
       cod_dg,
       cod_vert,
       nat_cabo,
       capac_cabo
from   cabo C,estacao E
where  E.cod_est = C.cod_est #

JANELsagre#2#JoinEstCabo#1#10#100#
CAMPOsagre#2#1#estacao#cod_est
```

```
SELECsagre#2#1#100#0#0#nat_cabo, sum(capac_cabo)#GROUP
>>BY (nat_cabo)##1^
```

Apêendice D

Execução do Benchmark Sagre

Apresentamos abaixo um trecho da execução do *benchmark* em uma máquina VAX 785 utilizando o SGBD Oracle.

```
DOC>sagre;3;S;1
DOC>#

COD_LANCE_CABO CONT_INICIAL CONT_FINAL PARES_MORTOS
-------------- ------------ ---------- ------------
           136          636       1039           95

ELAPSED: 0 00:00:01.05 CPU: 0:00:00.91
BUFIO: 5 DIRIO: 5 FAULTS: 318

DOC>sagre;9;S;1
DOC>#

    COD_SS     COD_EST        AREA
---------- ---------- ----------
         2          3       5179
        13          3      31266

ELAPSED:    0 00:00:00.78  CPU: 0:00:00.73
BUFIO: 6  DIRIO: 1  FAULTS: 46

DOC>sagre;3;S;1
DOC>#
```

```
COD_LANCE_CABO CONT_INICIAL CONT_FINAL PARES_MORTOS
-------------- ------------ ---------- ------------
            45          977       1391          185

ELAPSED:    0 00:00:00.38  CPU: 0:00:00.38
BUFIO: 5  DIRIO: 0  FAULTS: 5

DOC>sagre;4;S;1
DOC>#

COD_LANCE_CABO    COD_EST     COD_DG   COD_VERT   COD_CABO
-------------- ---------- ---------- ---------- ----------
            54          8          6          1         99

ELAPSED:    0 00:00:02.59  CPU: 0:00:01.83
BUFIO: 6  DIRIO: 5  FAULTS: 672
DOC>sagre;2;S;1

DOC>#

NAT_ SUM(CAPAC_CABO)
---- ---------------
X1XX            3395
X2XX            1236
X3XX            4831

ELAPSED:    0 00:00:01.00  CPU: 0:00:00.92
BUFIO: 7  DIRIO: 3  FAULTS: 64
```