FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA – UNIFOR

Dorotéa Karine Dias Pitombeira

Uma Arquitetura Eficiente para Armazenamento, Gerenciamento e Acesso a Dados em Dispositivos Móveis com Recursos Computacionais Limitados

Fortaleza
2006

FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA – UNIFOR

Dorotéa Karine Dias Pitombeira

Uma Arquitetura Eficiente para Armazenamento, Gerenciamento e Acesso a Dados em Dispositivos Móveis com Recursos Computacionais Limitados

Dissertação apresentada ao Curso de Mestrado em Informática Aplicada da Universidade de Fortaleza como parte dos requisitos necessários para a obtenção do Título de Mestre em Ciência da Computação.

Orientador: Prof. Dr.-Ing Ângelo Roncalli Alencar Brayner

Fortaleza
2006

Dorotéa Karine Dias Pitombeira

Uma Arquitetura Eficiente para Armazenamento, Gerenciamento e Acesso a Dados em Dispositivos Móveis com Recursos Computacionais Limitados

Data de Aprovação: _____

Banca Examinadora:

_____

Prof. Ângelo Roncalli Alencar Brayner, Dr.-Ing.
(Presidente da Banca)

_____

Prof. Javam de Castro Machado, Docteur
(Universidade Federal do Ceará – UFC)

_____

Prof. Nabor das Chagas Mendonça, Ph.D.
(Universidade de Fortaleza - UNIFOR)

PITOMBEIRA, DOROTÉA KARINE DIAS

Uma Arquitetura Eficiente para Armazenamento, Gerenciamento e Acesso a Dados em Dispositivos Móveis com Recursos Computacionais Limitados [Fortaleza] 2006

xvii, 120p., 29,7 cm (MIA/UNIFOR, M.Sc., Ciência da Computação, 2006)

Dissertação de Mestrado – Universidade de Fortaleza, MIA

1. Banco de Dados
2. Banco de Dados Móveis
3. Armazenamento de Dados

I. MIA/UNIFOR II. TÍTULO (série)

Dedico este trabalho aos meus pais,
Maria Dias e Alberto Gomes, por todo
amor e apoio que sempre me deram.

Agradecimentos

A Deus, por ser estar sempre presente em minha vida me confortando nos momentos difíceis, me ajudando a superar os problemas e me dando forças para seguir sempre em frente.

Aos meus pais, pela dedicação, amor, carinho e educação que sempre me deram e que foram fundamentais para a minha formação pessoal e profissional. Vocês são a minha referência, a vocês eu devo a minha vida, a minha história e o que hoje sou.

A toda a minha família, pelo apoio e incentivo que me deram durante a realização deste trabalho.

Ao Prof. Dr. Ângelo Brayner, pela competência, seriedade e compromisso sempre demonstrados, e pela oportunidade de poder participar de um de seus projetos de pesquisas, local onde surgiu o tema deste trabalho. Seu apoio nos momentos de dificuldade e sua orientação durante todo o desenvolvimento desta dissertação foram essenciais para a conclusão deste trabalho.

Aos professores Dr. Nabor das Chagas Mendonça e Dr. Javam de Castro Machado, pela presença na banca examinadora e por suas importantes considerações que resultaram no aperfeiçoamento deste trabalho.

Aos professores do Mestrado em Informática Aplicada (MIA) da Universidade de Fortaleza, pela contribuição direta ou indireta que deram à realização desse trabalho, em especial aos professores Dr. Nabor das Chagas Mendonça e Dr. Arnaldo Dias Belchior.

Ao Prof. Dr. Plácido Pinheiro, coordenador do MIA, pelo apoio demonstrado durante os momentos difíceis desta caminhada.

À Tânia, secretária do MIA, pela presteza e atenção sempre demonstradas.

Ao Centro de Aperfeiçoamento de Professores do Ensino Superior - CAPES, pela concessão de bolsa de estudo através do programa PROSUP.

Ao projeto Celestica/Palm financiado pelas empresas Celestica e Palm do Brasil e apoiado pela Lei de Informática, pela oportunidade de poder participar de pesquisas na área da computação móvel que resultaram na definição do tema desta dissertação. A infra-estrutura existente no projeto foi essencial para viabilizar a implementação e validação da proposta aqui apresentada.

Aos colegas do Projeto Celestica/Palm, pela contribuição na implementação do protótipo: Danielle Amorim, Magno Prudêncio, Marília Mendes, Ricardo Wagner, Thiago Leite, Wander Castro, Wandré Oliveira e Wendel Bezerra.

A todos os amigos da turma do Mestrado, pelo aprendizado que tive com discussões e trocas de idéias e conhecimento durante todo o curso. Também agradeço pelo apoio que recebi nos momentos mais difíceis, em especial aos amigos Rejane Pinheiro, Fernando Siqueira, Pascale Correia, Tatiana Monteiro, Lívia Nojoza, Ricardo Régis e Ricardo Wagner.

Por fim, a todas as pessoas que contribuíram direta ou indiretamente para a realização deste trabalho, meus sinceros agradecimentos.

Resumo da Dissertação apresentada ao MIA/UNIFOR como parte dos requisitos necessários para a obtenção do título de Mestre em Ciência da Computação

UMA ARQUITETURA EFICIENTE PARA ARMAZENAMENTO, GERENCIAMENTO E ACESSO A DADOS EM DISPOSITIVOS MÓVEIS COM RECURSOS COMPUTACIONAIS LIMITADOS

Dorotéa Karine Dias Pitombeira

Dezembro / 2006

Orientador: Dr.-Ing. Ângelo Roncalli Alencar Brayner
Programa: Ciências da Computação

A utilização de dispositivos móveis como *handhelds* e *Personal Digital Assistants* (PDAs) para acessar dados em qualquer lugar e a qualquer momento vem se tornando cada vez mais comum. Contudo, tais dispositivos apresentam recursos computacionais limitados. Portanto, a otimização de espaço utilizado em memória e a eficiência no acesso aos dados são fatores críticos. Esse trabalho apresenta uma estratégia eficiente para o armazenamento, o gerenciamento e o acesso aos dados em dispositivos móveis com recursos computacionais limitados. Para isso, foi definida e implementada uma estrutura de armazenamento baseado em colunas em que os dados de uma mesma coluna (ou atributo) são armazenados continuamente em memória. A estratégia implementada permite: (1) trabalhar com estruturas que manipulam os dados através de uma visão relacional; (2) criar e manter os esquemas de dados; (3) melhorar a performance de operações de busca com filtros verticais (projeções); (4) definir mecanismos para garantir a consistência dos dados através da manutenção de restrições de integridade; e (5) conseguir altas taxas de compressão de dados em dispositivos PDAs. Os resultados experimentais comprovam ganhos significativos de desempenho aliados à diminuição de espaço em memória.

Abstract of Thesis presented to MIA/UNIFOR as a partial fulfillment of the requirements for the degree of Master of Science Computer

AN EFFICIENT ARCHITECTURE FOR STORING, MANAGING AND ACCESSING DATABASE IN MOBILE DEVICES WITH LIMITED COMPUTATIONAL RESOURCES

Dorotéa Karine Dias Pitombeira

December / 2006

Advisor: Dr.-Ing. Ângelo Roncalli Alencar Brayner
Department: Computing Science

The use of mobile devices such as handhelds and Personal Digital Assistants (PDAs) to data access anywhere and any time is becoming more common. However, PDAs devices have limited computational resources. Therefore, reduction of memory space and efficiency in accessing data are critical issues. This work presents an efficient framework to data access, storage and management in mobile devices with limited computational resources. Thus, a column storage strategy was defined and implemented in which the values for each single column (or attribute) are stored contiguously in memory. This storage strategy provides: (1) data manipulation through relational view; (2) support for dealing with schema evolution; (3) optimization of queries with vertical filters (projections); (4) mechanism to guarantee the data consistency through to the maintenance of integrity constraints; and (5) storage strategy to increase the data compression rate in the PDAs devices. Experimental results show improvements of performance and reduction of required memory space to data storage.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API	*Application Programming Interface*
DDL	*Data Definition Language*
DML	*Data Manipulation Language*
EDGE	*Enhanced Data Rates For Global Evolution*
ERB	Estação Rádio Base
GPRS	*General Packet Radio Service*
HTML	*Hyper Text Markup Language*
IEEE	*The Institute of Electrical and Electronics Engineers, Inc.*
IP	*Internet Protocol*
ISM	*Industrial, Scientific, and Medical*
LAN	*Local Area Networks*
PDA	*Personal Digital Assistants*
PDB	*Palm Database*
PDM	*Palm Database Manager*
PQA	*Palm Query Application*
PRC	*Palm Resource*
RAM	*Random Access Memory*
ROM	*Read Only Memory*
SGBD	Sistema Gerenciador de Banco de Dados
SIG	*Special Interest Group*
SQL	*Structured Query Language*

TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
WAP	*Wireless Application Protocol*
Wi-Fi	*Wireless Fidelity*

Capítulo 1

Introdução

Este capítulo apresenta as principais questões que motivaram a realização do presente trabalho, assim como seus objetivos e sua organização.

1.1 Motivação

A utilização de dispositivos móveis como *notebooks*, *handhelds* e PDAs (*Personal Digital Assistants*) para acessar dados de qualquer lugar e a qualquer momento vem se tornando cada vez mais comum no cenário atual. A junção das tecnologias de comunicação sem fio e equipamentos portáteis com diversas funcionalidades vem consolidando o paradigma da computação móvel. Neste ambiente, os usuários utilizam dispositivos portáteis para, através de uma comunicação sem fio, acessarem outros dispositivos, móveis ou não, realizando transmissões, consultas e atualizações de dados independentes de sua localização. Juntamente com a computação móvel surgiram vários novos campos de trabalhos que necessitam ser explorados, gerando grande demanda em pesquisa e desenvolvimento nesta área.

O aumento na utilização destes dispositivos móveis tem representado um grande impulso no desenvolvimento de novas aplicações para o segmento da computação móvel. Isto se deve ao fato de que, atualmente, a possibilidade de gerenciar e acessar dados de qualquer lugar e a qualquer momento tornou-se um grande diferencial dentro do mundo globalizado. Em um curto prazo, estruturas sem fio e com fio permitirão uma conectividade entre os mais diversos equipamentos.

Dispositivos do tipo PDAs apresentam recursos computacionais limitados, com restrição de energia, reduzida área de memória e baixo poder de processamento, quando comparados ao ambiente *desktop* e até mesmo *laptops* e *notebooks*. Para tais dispositivos, o armazenamento eficiente, a facilidade de acesso aos dados, a otimização no processamento e a

redução de espaço utilizado em memória são fatores importantes que devem ser levados em conta para disponibilizar um ambiente eficaz de utilização. Considerando estes aspectos, a ausência de um mecanismo de armazenamento de dados bem estruturado nestes ambientes é uma das suas principais desvantagens. Com isto, passa a existir uma dependência direta entre a estrutura física de armazenamento dos dados e as aplicações. A arquitetura nativa destes ambientes não oferece um nível de abstração dos dados, de forma que tais estruturas assemelham-se a simples arquivos em ambientes convencionais.

Para disponibilizar um ambiente com uma maior facilidade de acesso aos dados aliado à possibilidade de redução no custo de utilização de memória, foi definida uma estratégia de armazenamento, acesso e gerenciamento dos dados que permite trabalhar com uma camada de abstração através de visão relacional e aplicar técnicas de compressão de dados visando reduzir espaço em memória utilizado para o armazenamento.

1.2 Objetivos

O objetivo deste trabalho é propor uma estratégia de armazenamento de dados para dispositivos portáteis do tipo PDA que garanta que os dados em tais dispositivos sejam armazenados de forma eficiente, com facilidade de acesso através de uma visão relacional dos dados aliada à possibilidade de redução dos custos de armazenamento. Esta estrutura está associada a um mecanismo gerenciamento que disponibiliza as características essenciais de um sistema gerenciador de banco de dados convencional. Nesta proposta, as técnicas de armazenamento utilizadas levam em consideração as particularidades destes dispositivos como o sistema de arquivo existente e as limitações das estruturas físicas de armazenamento.

Para tanto, foram desenvolvidas uma biblioteca com um conjunto de funções que definem a estrutura de armazenamento proposta e uma ferramenta que disponibiliza as funcionalidades que viabilizam a utilização da arquitetura pelos usuários. Para demonstrar a eficácia da estratégia são apresentados os resultados dos experimentos realizados.

1.3 Metodologia

Para a realização deste trabalho, foi feito um estudo bibliográfico na literatura, visando o entendimento teórico necessário sobre o armazenamento de dados em dispositivos móveis do tipo PDAs (*palmtops* e *handhelds*).

Foram analisadas as formas existentes de armazenamento de dados nestes dispositivos. As pesquisas concentraram-se nas principais dificuldades e problemas existentes nestas estruturas, bem como nos mecanismos envolvidos para gerenciá-las. Percebeu-se que propriedades essenciais para a boa gerência dos dados não eram facilmente disponibilizadas. A partir daí, foi proposta uma nova estrutura de armazenamento e gerenciamento de dados em ambientes com recursos computacionais limitados. Essa proposta foi refinada ao longo dos experimentos realizados de forma que foram acrescentados outros itens considerados importantes como garantia da consistência dos dados, suporte à mobilidade e otimização do processamento de consultas e da compactação dos dados.

1.4 Organização do trabalho

Este trabalho está organizado nos capítulos descritos a seguir.

No Capítulo 2, **Ambientes Computacionais para Dispositivos Móveis**, são apresentados alguns aspectos relacionados à computação móvel, bem como características básicas de um ambiente com recursos computacionais limitados como dispositivos portáteis do tipo PDAs (*palmtops* e *handhelds*). Por fim, são descritos alguns dos sistemas operacionais desenvolvidos especificamente para tais equipamentos.

No Capítulo 3**, Sistemas de Bancos de Dados para Dispositivos PDAs**, discorre-se sobre os principais sistemas gerenciadores de bancos de dados móveis existentes, atualmente, no mercado e desenvolvidos especificamente para dispositivos móveis. São descritas as principais características, funcionalidades e limitações de cada um deles.

No Capítulo 4, **Uma Estratégia Eficiente para Armazenamento e Acesso a Dados em Dispositivos PDAs**, apresenta-se a proposta com a contribuição deste trabalho: uma estratégia de armazenamento, acesso e gerência de dados em dispositivos portáteis com recursos computacionais limitados. A estrutura física de armazenamento dos dados e metadados, o mecanismo de acesso aos dados e as principais funcionalidades implementadas pela arquitetura são apresentadas do decorrer deste capítulo. A biblioteca DBLib também é descrita juntamente com a estrutura de gerenciamento de dados que dá o suporte necessário para viabilizar a estratégia definida. Também é apresentada a importância da compressão de dados nestes ambientes e os benefícios relacionados a este aspecto que são oferecidos pela arquitetura proposta. São descritas as técnicas de compressão dos dados definidas como uma

extensão baseada na estrutura de armazenamento proposta neste trabalho. Por fim, é apresentado um relato sobre os trabalhos relacionados.

No Capítulo 5, **Estudo de Caso: A Ferramenta *Palm Database Manager***, descreve-se o estudo de caso realizado. São apresentadas a arquitetura e a ferramenta PDM, descrevendo todos os componentes definidos na arquitetura e as principais funcionalidades existentes na ferramenta. Também são descritos os experimentos realizados utilizando a arquitetura PDM com o objetivo de avaliar e validar a estrutura de armazenamento de dados proposta no Capítulo 4 e apresentar os principais resultados obtidos.

No Capítulo 6, **Conclusão**, são apresentadas as considerações finais deste trabalho expondo suas contribuições e perspectivas futuras sobre o assunto apresentado. Também são descritas as publicações obtidas com esta proposta.

No Apêndice A são descrita as funções da API *Database Library* (DBLib).

Capítulo 2

Ambientes Computacionais Para Dispositivos Móveis

Este capítulo apresenta algumas características importantes dos ambientes de computação móvel e ambientes existentes para dispositivos portáteis. Além disto, também é apresentada uma revisão bibliográfica sobre os sistemas operacionais para dispositivos móveis com recursos computacionais limitados do tipo PDA.

2.1 Introdução

Diante da evolução computacional, as pessoas têm, atualmente, a possibilidade de acessar informações em qualquer lugar e a qualquer momento. Com um crescimento estimado entre 30% e 50% por ano, o público de usuários de aplicações executadas em dispositivos móveis como PDAs (*palmtops* e *handhelds*) vêm se tornando cada vez mais importante.

Quando comparados a equipamentos com mais recursos como *laptops* e *notebooks*, os *handhelds* possuem um grau de praticidade ainda maior, que começa desde seu tamanho e peso, até a facilidade no acesso e uso. Hoje, estes dispositivos são bem mais do que simples agendas eletrônicas, eles são equipamentos que atingem um alto grau de tecnologia embutida para disponibilizar para os seus usuários, além das funções básicas de cadastro de endereços, de tarefas e bloco de notas, outras funcionalidades como acesso a Internet, sistemas de vendas, sistemas de cadastro, filmadora, telefone celular e máquina fotográfica. Todas estas funcionalidades podem ser encontradas em um único dispositivo PDA e esta característica torna esses tipos de equipamentos cada vez mais adequados à necessidade da vida moderna. Com este crescente avanço na tecnologia de equipamentos portáteis de pequeno porte e com a diminuição dos custos destes dispositivos, a computação móvel vem se tornando cada vez mais presente no cotidiano das pessoas.

Estes dispositivos são computadores que podem ser levados para qualquer lugar, permitindo uma maior agilidade e oferecendo condições para atender profissionais em movimento que necessitam de rapidez, facilidade de uso e segurança no acesso a informações corporativas e pessoais. Para aqueles que consomem grande parte do seu tempo trabalhando remotamente, estes equipamentos são bastante adequados por serem versáteis, dedicados, multifuncionais e de uso genérico.

Por este motivo, esse segmento vem crescendo de forma constante e chamando a atenção das empresas de desenvolvimentos de aplicativos voltados para o ambiente móvel. Várias linhas de pesquisas têm surgido visando solucionar problemas decorrentes de características intrínsecas destes ambientes como a limitação dos recursos computacionais disponíveis nestes dispositivos portáteis.

Este capítulo aborda vários aspectos relacionados à computação móvel, tecnologia de comunicação sem fio, limitações de *hardware* para dispositivos PDAs (memória e energia) e sincronização de dados. Ainda são apresentadas algumas características relacionadas aos sistemas operacionais desenvolvidos especialmente para estes tipos os dispositivos portáteis, bem como são abordados aspectos referentes à estrutura de armazenamento e gerenciamento de dados nestes ambientes.

Na seção 2.2 são apresentados os principais conceitos existentes na área da computação móvel. Na seção 2.3 são descritas algumas características de um ambiente móvel com recursos computacionais limitados como sincronização e política de economia de energia. Na seção 2.4 são abordados alguns sistemas operacionais existentes para dispositivos portáteis do tipo PDA.

2.2 Características de um Ambiente de Computação Móvel

A computação móvel é uma área relativamente recente da computação e tem como principal aspecto permitir que o usuário tenha acesso a dados e serviços remotamente, podendo comunicar-se com outros dispositivos móveis ou fixos independentemente de onde estão situados ou de mudanças de localização [18]. Para viabilizar este ambiente móvel, o modelo computacional tradicional precisou ser modificado com o objetivo de integrar as tecnologias de comunicação sem fio e de computadores móveis.

Os elementos que, normalmente, compõem o ambiente móvel são elementos relacionados ao *software* como os agentes móveis, aplicações e serviços que suportam a mobilidade; os elementos de *hardware* compostos pelos dispositivos móveis como PDAs, telefones celulares e *notebooks*; e os usuários que interagem com este ambiente acessando serviços e solicitando informações.

As tecnologias de computação móvel abrangem tanto os dispositivos móveis considerados de pequeno porte como PDAs (*handhelds* e *palmtops*), quanto aqueles com maior poder computacional como os *laptops* e *notebooks.* Mais recentemente, surgiu no mercado um outro tipo de dispositivo portátil: os *smartphones,* denominados como telefones inteligentes que comportam as características da computação móvel existentes nos PDAs associadas às funcionalidades da telefonia celular. Todos estes dispositivos devem disponibilizar condições para que os usuários possam acessar serviços computacionais do ambiente móvel do qual faz parte em qualquer lugar e a qualquer momento.

2.2.1 Aspectos Relacionados à Computação Móvel

A computação móvel possui um conjunto de características próprias que a difere do ambiente tradicional. Tais aspectos apresentados a seguir e abordados em [10], [14], [15], [34] e [36] precisam ser considerados ao projetar novas aplicações e soluções para o ambiente com suporte à mobilidade.

2.2.1.1 Mobilidade

A mobilidade introduz problemas e desafios não existentes em ambientes onde existe apenas unidades fixas [18]. Os problemas existentes na computação móvel possuem particularidades que lhes são inerentes, tais como a taxa de transmissão da comunicação sem fio; a localização da unidade móvel; a segurança através de acesso remoto ou uma possível desconexão ou cancelamento de transações por motivos externos à transação, como a descarga de bateria da unidade móvel [8][9]. Por conta disto, a computação móvel surge com a necessidade de tratar aspectos que até então não precisavam ser considerados na computação tradicional com computadores fixos.

Outro fator a ser observado em relação à mobilidade é o custo para localizar uma unidade móvel, o que contribui de forma significativa para o custo de cada comunicação.

Dessa forma, vários aspectos computacionais precisam ser reavaliados para serem considerados satisfatórios quando inseridos no ambiente com suporte à mobilidade.

2.2.1.2 Handoff

Um ambiente com suporte à mobilidade é composto por unidades móveis agrupadas em células. Cada célula representa uma determinada região coberta por um serviço de comunicação sem fio. A movimentação destas unidades móveis dentro de uma célula de cobertura ou entre células vizinhas durante a execução de aplicações gera um processo chamado *handoff*. Este processo acontece entre duas estações base adjacentes e garante uma transação enquanto os usuários se movimentam de uma célula para outra, mantendo a conectividade de uma unidade móvel com a rede fixa [33].

Isto implica que uma estação base pode detectar qualquer entrada e saída de usuários da sua célula. Durante o processo de *handoff*, as estações base são responsáveis pela comunicação sem fio das unidades móveis, de forma que este processo é transparente para o usuário.

2.2.1.3 Desconexão

Em ambientes com suporte à mobilidade as desconexões são freqüentes [2] e podem ser caracterizadas como voluntárias ou forçadas. Uma desconexão é voluntária quando o usuário não deseja acessar a rede visando diminuir o custo da comunicação, o consumo de energia ou o uso da largura de banda. Uma desconexão é forçada quando o usuário entra numa região onde não há cobertura ou não existe canal de comunicação.

Também devido à mobilidade, a qualidade do canal de comunicação tende a variar e, como conseqüência disto, protocolos de comunicação, sistemas operacionais e aplicativos devem estar preparados para lidar com flutuações na qualidade da conexão.

2.2.1.4 Gerenciamento de Energia

Unidades móveis dependem de baterias para funcionar. Dessa forma, o gerenciamento de energia é um problema importante que deve ser considerado quando se trata de ambientes móveis. Nestes equipamentos a energia é um recurso limitado de forma que o seu consumo deve ser otimizado [2]. A otimização de energia está diretamente relacionada à otimização das tarefas que são realizadas para atender a uma solicitação do usuário.

A velocidade de processamento, o envio e o recebimento de dados durante um acesso remoto e o custo de armazenamento destes dados em memória em termos de energia devem ser fatores importantes a serem considerados sob o aspecto de organização e acesso aos dados [14].

2.2.2 Tecnologias de Comunicação Sem Fio

Considerando que a computação móvel tem como princípio básico a mobilidade, a utilização da comunicação sem fio é um dos aspectos essenciais que dão suporte a esta área, pois este tipo de comunicação elimina a necessidade do usuário manter-se conectado a uma rede com infra-estrutura fixa. Isto significa a possibilidade de acessar os dados e aplicativos de forma remota através de algum tipo de comunicação sem fio entre os dispositivos ou uma estação móvel. Assim, a comunicação sem fio é a base para a existência de cenários envolvendo mobilidade.

As redes de comunicação sem fio precisam estar disponíveis em todos os lugares, pois devem garantir a conectividade entre dispositivos independente de sua localização física. São consideradas tecnologias da comunicação sem fio utilizadas pela computação móvel o *Bluetooth*, WI-FI e a telefonia celular que são descritos a seguir.

2.2.2.1 Bluetooth

O *Bluetooth* [3] é uma tecnologia que possibilita incluir um sistema de comunicação sem fio em um único *chip* para ser embutido em telefones celulares, computadores móveis, PDAs e outros dispositivos portáteis possibilitando a comunicação entre eles. O *Bluetooth* é um padrão proposto pelo *Bluetooth* SIG (*Special Interest Group*) e opera na faixa ISM (*Industrial, Scientific, and Medical*) de 2,4 GHz. Este padrão permite a conexão entre diferentes tipos de dispositivos com baixo poder de processamento e pouca energia, a curtas distâncias, possibilitando a formação de redes *ad hoc*.

A estrutura básica de comunicação no *Bluetooth* é chamada de *piconet* [17]. Uma *piconet* é uma rede onde um nó central, chamado de mestre, comunica-se ponto-a-ponto com os outros nós, chamados de escravos, formando uma topologia estrela. Os dispositivos alternam-se nas funções de mestre e escravo. As *piconets* formam pequenas redes pessoais, conhecidas como PAN (*Personal Area Network*).

2.2.2.2 Wi-Fi

Uma outra tecnologia de comunicação são as LANs (*Local Area Networks*) sem fio ou Wi-Fi (*Wireless Fidelity*), padronizadas como IEEE (*The Institute of Electrical and Electronics Engineers, Inc*.) 802.11.

De acordo com [17], a especificação define uma camada de acesso ao meio, camada MAC (*Media Access Control*), e diferentes camadas físicas, tornando possível acessar o meio de três formas: FHSS (*Frequency Hopping Spread Spectrum*), DSSS (*Direct Sequence Spread Spectrum*) e infravermelho. O padrão 802.11 é chamado muitas vezes de "*Ethernet* sem fio" por ser uma extensão natural do padrão *Ethernet* (IEEE 802.3).

2.2.2.3 Redes Celulares

A tecnologia mais popular de comunicação sem fio de longa distância é formada pelos sistemas celulares. Atualmente, além da comunicação por voz, a telefonia celular permite a transmissão de dados a longas distâncias para usuários dispersos geograficamente.

De acordo com [18], o nome sistema móvel celular (SMC) tem origem em sua estrutura de células. Uma célula é uma área geográfica assistida por um transmissor de baixa potência, uma ERB (Estação Rádio Base). Uma ERB é composta por uma ou mais antenas fixas, instaladas em torres que têm como objetivo atender a demanda originada pela unidade móvel dentro de sua área de cobertura. A unidade móvel é o equipamento manipulado pelo usuário do SMC.

Os sistemas celulares implementaram serviços específicos para permitir a transmissão de dados, tais como GPRS (*General Packet Radio Service*) e EDGE (*Enhanced Data Rates For Global Evolution*).

O WAP (*Wireless Application Protocol*) é um protocolo de transmissão de dados na Internet para dispositivos móveis com comunicações sem fio, sendo independente do dispositivo e do sistema celular utilizado [34]. O modelo de comunicação WAP é similar ao modelo de comunicação Web, sendo, atualmente, um protocolo bastante utilizado para transmissão de dados na telefonia celular [18].

2.3 Características de um Ambiente Móvel com Recursos Computacionais Limitados

Apesar de cada vez mais compactos e com mais recursos, os dispositivos móveis do tipo PDAs ainda possuem certas limitações inerentes as suas características. As restrições de energia, reduzido poder de processamento e de armazenamento de dados e taxas de transmissão geralmente menores do que as das redes fixas estão entre as principais limitações existentes.

2.3.1.1 Armazenamento de Dados em Memória

Os dispositivos portáteis do tipo PDAs trabalham com uma concepção diferente no que diz respeito à forma de armazenamento de dados, uma vez que tais equipamentos não possuem meios magnéticos ou ópticos de armazenamento como componente padrão do dispositivo. Os dados são armazenados em memória RAM (*Random Access Memory*) em quantidade bastante inferior daquelas encontradas em ambientes *desktop*, não sendo recomendado que tais dispositivos precisem gerenciar grandes quantidades de dados. Isso é indicado para não prejudicar o poder de processamento.

Portanto, é importante considerar que nestes ambientes a otimização na utilização de espaço de memória e a eficiência no acesso aos dados são fatores que estão diretamente relacionados a um bom desempenho destes dispositivos e implicam em uma melhor performance na execução das tarefas.

É possível estender o armazenamento de dados através da utilização de cartões de memória. Tais recursos permitem expandir a capacidade de armazenamento limitada destes dispositivos. São exemplos de cartões de memória: *Secure Digital* (SD), *Compact Flash* (CF) e *Memory Stick* (MS). Através destes cartões de memória removíveis é possível realizar a transferência de dados de um dispositivo móvel para outro ou diretamente para um computador *desktop*. Apesar de expandir a capacidade de armazenamento do dispositivo, tais cartões são recursos adicionais que são integrados ao equipamento, de forma que não fazem parte da configuração padrão do dispositivo PDA. Por este motivo, não é possível considerar este aspecto com uma solução para a limitação de memória existente nestes ambientes, já que nem sempre o usuário tem interesse em adquirir cartões de memória para serem integrados ao dispositivo PDA.

Associado à otimização do armazenamento, também deve ser considerada a melhor forma de acesso e gerência dos dados gravados em memória. A eficácia na manipulação dos dados é o que permite uma melhor performance na execução das tarefas realizadas a partir de tais equipamentos. Dessa forma, esta gerência deve garantir alguns aspectos básicos referentes ao armazenamento eficiente de dados como a consistência dos dados, facilidade de acesso e melhor manutenibilidade das estruturas de armazenamento (esquemas).

Todos estes aspectos devem ser considerados para que as vantagens oferecidas pelos dispositivos portáteis possam estar disponíveis para o usuário em sua totalidade. Isto permite que tais equipamentos consigam atingir seu principal objetivo: oferecer acesso aos dados de forma eficiente a qualquer hora e lugar. Para tanto, faz-se necessário dispor de boas práticas para armazenar, gerenciar e recuperar os dados.

2.3.1.2 Política de Economia na Utilização de Energia

Um outro aspecto importante que deve ser considerado é a necessidade de otimizar o processamento visando atingir uma boa política de economia na utilização de energia do dispositivo. As fontes de energia como as baterias recarregáveis e as pilhas possuem um tempo de vida útil limitado que, de acordo com a tecnologia, oeferecem tempos de utilização que variam de algumas horas a alguns dias. Assim, para economizar energia, muitos equipamentos utilizam a estratégia de ficarem em estado de espera quando não estão em uso. Para dispositivos de pequeno porte, existe também o aspecto relacionado ao tamanho físico, pois quanto menor o equipamento, menor será o tamanho físico da sua fonte de energia e, conseqüentemente, maior será a probabilidade de possuir menor quantidade de energia armazenável.

Para contornar alguns dos problemas relacionados à fonte de energia, é importante focar na boa forma de utilização desta, uma vez que é necessário que o dispositivo tenha sempre a quantidade mínima de energia suficiente para não perder os dados existentes na memória de armazenamento temporário. Uma vez carregado, os dispositivos do tipo PDAs adquirem a energia necessária para sua operacionalização, porém quando esta energia é totalmente utilizada, os dispositivos perdem todos os dados armazenados nesta memória. Por isso, é extremamente importante a recarga de energia antes que a bateria acabe e os dados armazenados sejam completamente perdidos [27].

Para evitar este tipo de acontecimento, tais dispositivos possuem a capacidade de sincronizar o conteúdo de sua memória para um computador de mesa. Isso permite que o usuário possa manter uma cópia de segurança dos dados existentes no dispositivo móvel na memória secundária de um computador *desktop*. Caso existam cartões de memória, também é possível manter uma cópia dos dados nestes cartões para evitar a perda total dos dados armazenados na memória interna do dispositivo.

2.3.1.3 Sincronização

O processo de sincronização funciona de modo a assegurar que, ao final desta operação, tanto o dispositivo móvel quanto o outro computador fixo ou móvel possuam os mesmos dados. Tal procedimento pode ser realizado através de alguns tipos de comunicação disponíveis no dispositivo portátil como: a utilização de cabos seriais e USB, infravermelho e *Bluetooth* para a troca de dados entre dispositivos portáteis ou entre um dispositivo portátil e um computador *desktop*. Também é possível realizar comunicação através de uma rede sem fio ou Wi-Fi para permitir que o dispositivo móvel possa acessar uma unidade fixa a partir da rede local. Uma outra forma de comunicação existente é através de uma rede de telefonia celular, que permite enviar e receber dados a longas distâncias.

2.3.1.4 Usabilidade

A praticidade e a facilidade de uso dos dispositivos móveis existentes atualmente no mercado são características essenciais que os fazem tão populares. Para dar suporte a todos os aspectos descritos anteriormente deve existir um sistema operacional específico para tais dispositivos capaz de fornecer um ambiente com alto grau de usabilidade, que gerencie eficientemente o consumo de energia, o armazenamento e processamento de dados, bem como as características específicas de *hardware*. Este ambiente deve fornecer as condições necessárias para que tais equipamentos consigam efetivamente atingir os objetivos a que se propõem, disponibilizando o máximo de funcionalidades que um computador portátil possui associado à qualidade de acesso e facilidade de uso.

Os sistemas operacionais para dispositivos móveis de pequeno porte disponibilizam o aspecto usabilidade através da utilização de telas sensíveis ao toque, combinadas com a utilização de canetas especiais, como um dos meios de entrada de informação. Isso facilita bastante a utilização do equipamento pelo usuário.

2.4 Sistemas Operacionais para Dispositivos PDAs

São exemplos de sistemas operacionais que foram desenvolvidos especificamente para dispositivos móveis com recursos computacionais limitados o Palm OS e o Symbian OS. O Windows Mobile e o Linux OS possuem versões desenvolvidas especificamente para dispositivos portáteis de pequeno porte. Nesta seção, são descritos alguns dos sistemas operacionais para os dispositivos PDAs, como também suas principais características e arquitetura de armazenamento e acesso aos dados.

Os sistemas operacionais desenvolvidos especificamente para os dispositivos portáteis do tipo PDAs (*palmtops* e *handhelds*) fazem parte de um segmento de ambientes que precisam considerar as características referentes às limitações computacionais. Assim, estes sistemas operacionais, geralmente, apresentam características como:

- Ocupam pouco espaço em memória;
- Possuem portabilidade visando adequar-se a tipos diferentes de *hardware;*
- Possuem aspectos relacionados à gestão de energia visando economizar bateria.

Nas seções que seguem, são apresentados dois sistemas operacionais: o Palm OS e o Windows Mobile. O Palm OS será abordado com maior ênfase e de forma mais aprofundada por se tratar do ambiente utilizado no estudo de caso deste trabalho. Sobre o Windows Mobile será apresentada apenas uma visão geral, pois não foram encontradas referências bibliográficas suficientes para detalhar melhor o sistema de arquivo e a estrutura de armazenamento de dados deste sistema operacional.

2.4.1 O Ambiente Palm OS

O sistema operacional Palm OS vem sendo desenvolvido pela Palm desde 1996 para disponibilizar uma plataforma que permitisse aos usuários trabalhar com aquela que se tornaria uma grande tendência dos anos que se sucederam: a computação móvel e todos os benefícios que esta área oferece. Atualmente, o ambiente Palm OS é compatível com vários tipos de *hardware* apresentando grande flexibilidade.

2.4.1.1 Arquitetura da memória no Palm OS

Em um sistema tradicional de arquivos, os dados, geralmente, são lidos do disco para o *buffer*, operações são executadas com esses dados no *buffer* e, em seguida, os dados

são gravados de volta no disco. A estrutura de armazenamento de dados persistentes em PDAs ocorre de forma semelhante a dos sistemas de arquivos tradicionais. Por não possuir disco (memória secundária), o dispositivo utiliza uma área da memória RAM para armazenar esses dados. A menos que existam cartões de extensão de memória, um dispositivo PDA não trabalha de acordo com o sistema tradicional de arquivos encontrados em ambientes *desktop*, ou seja, não existe memória não volátil (disco) disponível para armazenar dados do usuário.

A memória de um PDA reside em um módulo, denominado *card* no caso específico do ambiente Palm OS [27]. Esse módulo é uma unidade lógica usada pelo sistema operacional para definir áreas de memória do tipo RAM e ROM (*Read Only Memory*). A memória RAM é dividida em duas áreas lógicas: RAM dinâmica e RAM de armazenamento. A primeira é usada como área de trabalho para alocações dinâmicas de aplicações e variáveis globais do sistema, enquanto a segunda é utilizada para armazenar os dados que não são temporários [30].

A RAM dinâmica é utilizada de forma similar à memória RAM do ambiente *desktop*, pois armazena variáveis globais de aplicações e de sistema e disponibiliza espaço em memória para alocar aplicações correntes não exigindo a persistência desses dados após sua utilização [27]. A Figura 1 ilustra a estrutura de memória definida e utilizada pelo ambiente Palm OS.

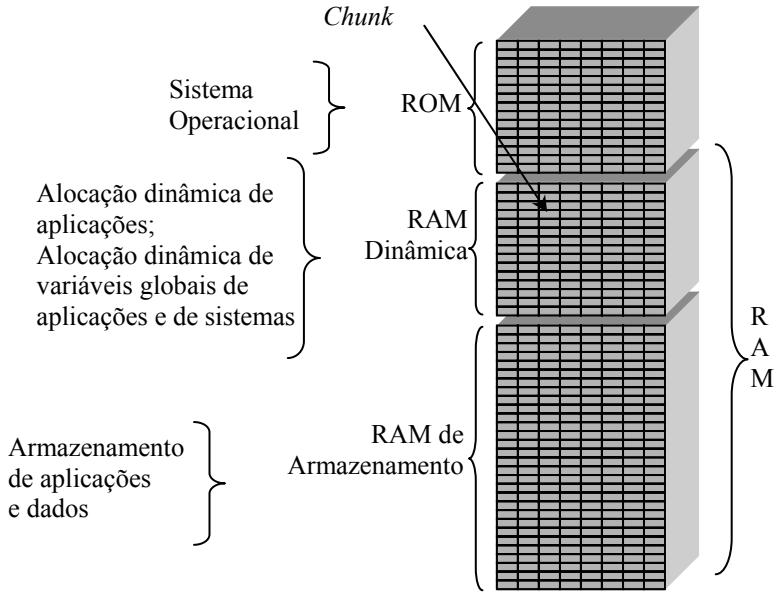

Figura 1 - Arquitetura da memória no ambiente Palm OS

A memória RAM de armazenamento preserva os dados e as aplicações de forma permanente, funcionando analogamente ao armazenamento secundário do ambiente *desktop*. Porém, esta área de armazenamento não é preservada quando ocorre uma operação de *hard reset*[1] ou quando a bateria do dispositivo acaba por completo. Quando isto acontece, os dados armazenados nesta memória são totalmente perdidos.

A ROM é uma memória não volátil, portanto as informações não serão perdidas mesmo que o dispositivo descarregue ou passe por uma operação de *hard reset*. A memória ROM armazena o sistema operacional do dispositivo, além de programas gravados pelo próprio fabricante, como agenda, calculadora e bloco de anotações.

Uma aplicação acessa os dados na área de armazenamento invocando as funções das bibliotecas do Palm OS: *Resource Manager* [29] ou *Data Manager* [29], de acordo com o tipo de dado a ser acessado. A biblioteca *Resource Manager* é responsável por gerenciar os dados referentes aos códigos fontes e *interfaces* de aplicações [30]. Já a biblioteca *Data Manager* é responsável por acessar os dados que serão utilizados por outras aplicações, dados do usuário como lista de endereço, cadastro de clientes, anotações ou documentos [30].

2.4.1.1.1 Heap e Chunk de Memória

De acordo com [27], um *heap* é uma área contígua de memória formada por pequenas unidades chamadas *chunks*. As APIs que possibilitam acessar os dados nestes ambientes consideram um *chunk* como a menor unidade de armazenamento de dados em memória. Dessa forma, todos os dados existentes no ambiente Palm OS são armazenados nestas unidades e o conjunto desses *chunks* armazena todos os dados existentes no dispositivo. Um *chunk* é uma área contígua de memória de 1 a 64 KB de tamanho [27]. A aplicação, ao definir uma nova unidade de memória, pode especificar se a alocação do novo *chunk* de memória será na memória RAM de armazenamento ou na RAM dinâmica.

Como já mencionado, a memória RAM dinâmica é usada como área de trabalho para alocações dinâmicas de dados, enquanto a memória RAM de armazenamento é utilizada para armazenar os dados que não são temporários. Toda área dinâmica da RAM é utilizada para implementar, geralmente, um único *heap* dinâmico, o restante de memória RAM não dedicado ao *heap* dinâmico é configurado como um ou mais *heaps* de armazenamento [30].

---

[1] *Hard reset* é uma operação que quando executada limpa toda a área da memória RAM dinâmica e RAM de armazenamento. Permanecem gravados no sistema apenas os dados da ROM.

As estruturas de memórias apresentadas podem ser acessadas pelas bibliotecas *Memory Manager* e *Data Manager*. A biblioteca *Memory Manager* é responsável pelo gerenciamento dos *chunks* de memória no *heap* dinâmico, enquanto a *Data Manager* gerencia os *chunks* de memória no *heap* de armazenamento [27]. A classificação de cada *heap* é dada de acordo com seu identificador (*heap* ID) que permite identificar um *heap* dentro do espaço de endereço de memória. Se este for igual a 0 (zero) ele será gerenciado pela *Memory Manager,* caso contrário pela *Data Manager*.

De acordo com [30], todos os dados do dispositivo são armazenados em *chunks*. Cada *chunk* reside em um *heap*. Os *heaps* da memória ROM contêm apenas *chunks* fixos, já os *heaps* da memória RAM podem ter *chunks* fixos ou móveis. Quando o *Memory Manager* aloca um *chunk* fixo, ele identifica o endereço daquele *chunk* em memória e o retorna para a aplicação. Por isso, *chunks* fixos não podem ser movidos, pois seu endereço permanece válido enquanto existir aquele *chunk*. Quando o *Memory Manager* aloca um *chunk* móvel, ele gera um endereço para aquele *chunk*, da mesma maneira que fez para o *chunk* fixo, porém armazena este endereço em uma tabela que é usada para localizar todos os *chunk* móveis no *heap.* A referência para este endereço do *chunk* móvel é conhecida como *handle*. O uso do *handle* permite ao *Memory Manager* mover *chunks* no *heap* sem invalidar qualquer referência de *chunk* que uma aplicação possa ter armazenado anteriormente. Quando uma aplicação utiliza *handles* para referenciar dados, apenas a tabela de endereços do *chunk* precisa ser atualizada pelo *Memory Manager*.

Uma aplicação bloqueia temporariamente o *chunk* quando precisa ler ou manipular o conteúdo desse *chunk* [30]. O processo de bloquear um *chunk* diz para o *Memory Manager* que aquele *chunk* de dados está definido como imóvel. Quando uma aplicação já não precisar do *chunk* de dados, o *handle* é desbloqueado para que possa ser utilizado novamente por outro processo. Todo *handle* só é válido até que o sistema seja reiniciado. Quando o sistema é totalmente desligado e ligado novamente, ele inicia todas as áreas de memória dinâmicas e carrega novamente as aplicações. Por isso, não se deve armazenar um *handle* em um registro de dados.

As funções da API *Memory Manager* são invocadas se for necessário alocar memória no *heap* dinâmico [27]. Caso seja necessário alocar memória no *heap* de armazenamento, as funções da *Data Manager* são chamadas. Essa decisão é tomada pelo sistema operacional de acordo com a especificação da aplicação, já que ela pode definir se a

alocação de um novo *chunk* de memória será no *heap* de armazenamento ou no *heap* dinâmico.

Cada *heap* é administrado de forma independente pela *Memory Manager*. Esta API é responsável pela manutenção da localização e tamanho de cada *chunk* de memória nas áreas de armazenamento volátil, não volátil e memória ROM [30]. Ela também permite a alocação, remoção, redimensionamento, bloqueio de *chunks* e a compactação de *heaps* quando estes se encontram fragmentados. Devido ao limite de recursos do dispositivo, as funções da *Memory Manager* precisam ser eficientes na utilização de poder de processamento e memória.

Todo *chunk* de memória usado para armazenamento de dados é um registro em um arquivo de dados implementado pela API *Data Manager* que tem como finalidade principal o gerenciamento dos dados do usuário que estão armazenados em registro de dados. Esta API utiliza as funções do *Memory Manager* para alocação, remoção e redimensionamento (até o limite do *chunk*) de registros de dados, disponibilizando funções para inserção, atualização, exclusão e busca dos dados que residem na memória RAM de armazenamento.

Conforme [30], no ambiente Palm OS, um arquivo de dados é uma lista de *chunks* de memória associada às informações de um determinado conjunto de dados de forma que todos esses *chunks* fazem parte de uma associação lógica. Um arquivo de dados, neste ambiente, é análogo a um arquivo em um sistema *desktop* convencional, exceto porque se localiza na memória RAM de armazenamento. De forma similar a um arquivo, as aplicações no Palm OS podem criar, remover, abrir e fechar os registros de dados. A RAM dinâmica é uma memória volátil, por isso ela não pode armazenar dados de forma permanente. Assim, os registros de dados, por necessitarem de armazenamento permanente, precisam ser alocados na memória RAM de armazenamento.

2.4.1.2 Estrutura de Armazenamento de Dados no Palm OS

O Palm OS trabalha com diferentes tipos de formatos de arquivos para armazenar seus dados: *Palm Database* (PDB), *Palm Query Application* (PQA) e o *Palm Resource* (PRC) [28]. Cada formato trabalha com tipos de dados distintos e são utilizados para finalidades específicas. No arquivo do tipo PDB os dados são armazenados como um conjunto de dados dispostos seqüencialmente em memória que são acessados por outras aplicações ou pelo

sistema operacional [31]. Os arquivos do tipo PRC possuem dados referentes às aplicações e têm a finalidade de armazenar recursos como código fontes e elementos gráficos de *interface* para usuário como imagens, fontes e *layout* [31]. Já os arquivos do tipo PQA armazenam aplicações que contêm *interfaces* em HTML que possibilitam acessar informações pela rede através da Internet [31]. Estes três tipos de arquivos possuem estruturas semelhantes de armazenamento, a diferença está nos tipos de dados que são armazenados por cada um deles.

Este trabalho irá focar-se nos arquivos que armazenam os registros de dados que são acessados por outras aplicações: no caso do ambiente Palm OS, os arquivos do tipo PDB. Apesar de serem definidos pelo nome *Palm Database* (Banco de dados Palm), tais arquivos não possuem propriedades de bancos de dados. A Palm não possui uma aplicação proprietária para fornecer funcionalidades de um sistema de banco de dados tradicional [35], ela disponibiliza a API *Data Manager* que apresentam um conjunto de funcionalidades que permitem manipular os dados armazenados em memória em forma de arquivos [29].

Os dados armazenados em um arquivo do tipo PDB estão organizados como uma coleção de registros de forma que outras aplicações possam acessar estes dados. Conforme definido em [28], além dos dados de aplicativos (ou de usuários), tais arquivos apresentam ainda os seguintes dados de controle que compõem seu cabeçalho, conforme ilustra Figura 2:

- O cabeçalho que descreve algumas características do arquivo fazendo referência ao bloco de informações da aplicação (*AppInfo*) e ao bloco de ordenação dos dados (*SortInfo*). Neste ponto, também existe uma lista de entrada dos registros de dados existentes no arquivo;
- *AppInfo c*orresponde ao bloco que armazena as informações específicas da aplicação podendo ser opcional. Neste bloco são armazenadas informações como data de criação da estrutura de armazenamento, data da sua última alteração e data da última sincronização;
- *SortInfo* corresponde ao bloco que contém a ordenação dos registros de dados. Ele também pode ser opcional;
- Registro de dados armazenado seqüencialmente e referenciado na lista de entrada dos registros.

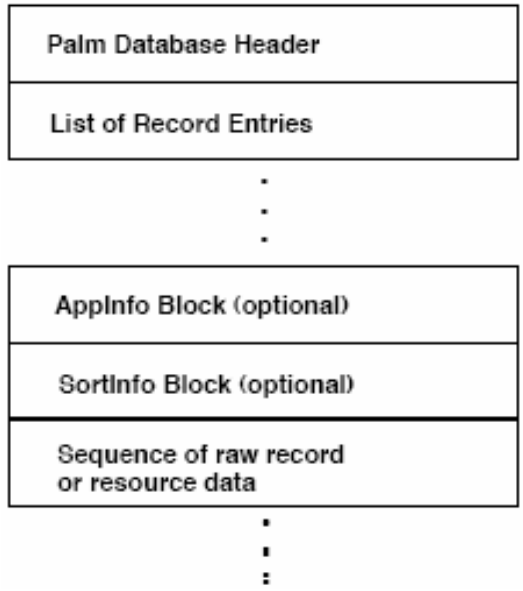

Figura 2 – Estrutura do arquivo de armazenamento dos dados [28]

De acordo com [28], um arquivo PDB apresenta uma estrutura, como a ilustrada na Figura 3, que mostra a representação lógica de um registro neste arquivo. O cabeçalho referencia as informações dos blocos *AppInfo* e *SortInfo* e termina com a lista de entrada dos registros que faz referência a cada registro armazenado no arquivo. Esta estrutura é semelhante para os arquivos do tipo PRC e PQA. Os dados que compõem os registros são armazenados em um bloco de dados contínuo de forma que a posição para o início de cada registro é especificada na lista de registros do cabeçalho do arquivo.

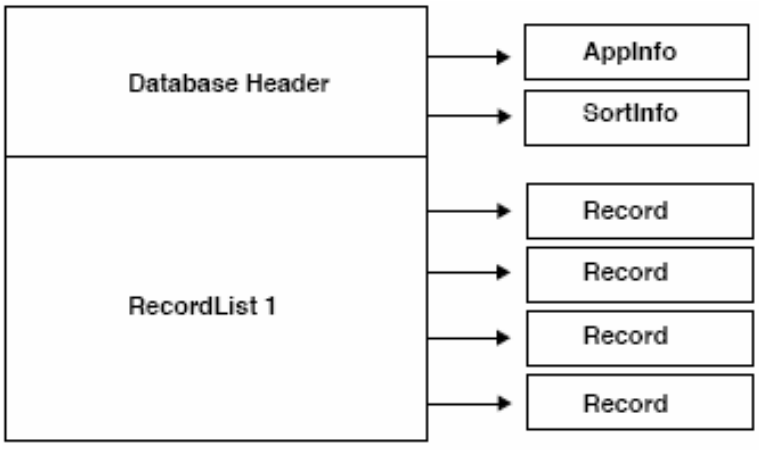

Figura 3 - Estrutura lógica de um arquivo de dados [28]

O armazenamento de dados no ambiente Palm OS trabalha com registros em arquivos não estruturados sem qualquer forma de abstração da estrutura de armazenamento física utilizada. A estrutura física pode ser de dois tipos: clássica ou estendida [27]. A

estrutura clássica é suportada por versões anteriores ao Palm OS v1.0. A estrutura estendida possui características adicionais à estrutura clássica como maior capacidade de armazenamento por registro e é suportada por todas as versões do Palm OS através da API *Data Manager*. Este último tipo pode armazenar até 64 KB em memória, porém não permite realizar operações mais complexas sobre os dados como, por exemplo, junção entre tabelas, pois não possui qualquer tipo de estrutura relacional.

A maioria das aplicações desenvolvidas para o ambiente Palm OS necessita conhecer a estrutura física de armazenamento em memória para realizar operações nos dados, atualização de estado dos registros, entre outros. Neste caso, tais aplicações são inteiramente responsáveis pela estruturação e pela interpretação do conteúdo de cada registro de dado. Esta estrutura dificulta a manutenção e o gerenciamento dos dados, pois a aplicação cliente fica sobrecarregada necessitando implementar controles adicionais que garantam a consistência dos dados utilizados.

2.4.2 O Ambiente Windows Mobile

O Windows Mobile é um sistema operacional da Microsoft desenvolvido para os dispositivos móveis do tipo PDA. Este ambiente apresenta funcionalidades como a realização de conexões sem fio com suporte nativo às tecnologias Wi-Fi e *Bluetooth*, e integração com o correio eletrônico para a sincronização de *e-mails* e agendas pessoais [19].

Os dados armazenados no sistema de arquivos do Windows Mobile são referenciados através de objetos de memória [19]. Estes objetos representam itens de dados do sistema de arquivos, do sistema de registros e dos demais dados armazenados em memória.

O Windows Mobile possui um banco de dados embutido que provê um conjunto de funcionalidades chamado Microsoft SQL Server 2005 Mobile Edition. Através do SQL Server Mobile é possível criar um objeto de memória que possui vários bancos de dados e inclui suporte para transações, acesso por múltiplos usuários e propriedades de sistemas de bancos de dados. Para tanto, é necessário adquirir a solução de banco de dados SQL Server Mobile que possui processador de consultas, suporte a sincronização e o gerenciamento de um maior volume de dados. A seção 3.2.1 apresenta o Microsoft SQL Server 2005 Mobile Edition de forma mais detalhada.

Não foi possível encontrar na literatura informações suficientes para apresentar de forma aprofundada o sistema de arquivos utilizado pelo Windows Mobile, porém é importante ressaltar a existência deste sistema operacional como um dos sistemas utilizados em dispositivos móveis.

Tão importante quanto o sistema operacional utilizado pelos dispositivos móveis são os sistemas de banco de dados específicos para estes equipamentos. No Capítulo 3, são abordados alguns dos SGBDs existentes para dispositivos PDAs.

Capítulo 3

Sistemas de Bancos de Dados Para Dispositivos PDAs

Este capítulo apresenta os principais sistemas de bancos de dados móveis existentes para dispositivos do tipo PDA.

3.1 Introdução

Atualmente, os grandes fabricantes de SGBDs tradicionais disponibilizam suas versões para o ambiente móvel. A comunicação sem fio e a sincronização de dados entre o dispositivo portátil e o servidor banco de dados da rede fixa são aspectos essenciais que precisam estar disponíveis para oferecer aos usuários as condições básicas de utilização da computação móvel. O gerenciamento eficiente destes dados também é uma característica importante que deve ser considerada em sistemas de banco de dados móveis voltados para dispositivos portáteis.

Com o objetivo de abordar algumas soluções existentes de SGBDs para ambientes móveis, na seção 3.2 são apresentadas algumas ferramentas que permitem trabalhar com banco de dados móveis. Aspectos específicos de cada fabricante também são abordados com o objetivo de mostrar as limitações de cada solução. Entre as ferramentas apresentadas estão Oracle 9i Lite, SQL Server Mobile, DB2 Everywhere e SQL Anywhere Studio. Na seção 3.3 é descrito um conjunto de critérios visando apresentar um estudo comparativo entre tais propostas.

3.2 Sistemas de Bancos de Dados Para Dispositivos PDAs

Os sistemas de banco de dados inseridos no contexto da mobilidade que acessam e gerenciam dados de forma distribuída são chamados de sistemas gerenciadores de banco de dados móveis e são considerados como uma variação dos sistemas de banco de dados distribuídos [47]. Os sistemas gerenciadores de banco de dados móveis podem estar presentes

não apenas em dispositivos móveis, mas também em computadores fixos que disponibilizam seus dados para os dispositivos móveis, desde que estejam integrados através de uma rede de comunicação sem fio.

Atualmente, existem alguns sistemas gerenciadores de banco de dados móveis específicos para dispositivos portáteis PDAs *(handhelds* e *palmtops*). Empresas como a Oracle, IBM, Microsoft e Sybase possuem versões de seus bancos de dados corporativos para dispositivos móveis.

3.2.1 Microsoft SQL Server 2005 Mobile Edition

O Microsoft SQL Server Mobile Edition (SQL Server Mobile) foi lançado em 2005 como uma nova versão do SQL Server 2000 Windows CE Edition. Novas funcionalidades de sincronização, mecanismo de armazenamento de dados, processamento de consultas e integração com outros produtos Microsoft foram incluídos nesta nova versão.

Conforme descrito em [20], a revisão do mecanismo de armazenamento de dados ocasionou a inclusão de características como o gerenciamento de transações e o uso de bloqueios para controlar a concorrência no acesso de usuário; a garantia da atomicidade, consistência, isolamento e durabilidade das transações; a criação de estrutura de índices; o suporte a integridade referencial e a criptografia de dados; e o gerenciamento de buffer e operações de I/O nos arquivos físicos. O processador de consultas passou a trabalhar com sentenças DML (*Data Manipulation Language*) permitindo a utilização de junções com `Order by` e das cláusulas `Group by` e `Distinct`.

A arquitetura do SQL Server Mobile é composta pela camada de desenvolvimento, cliente e servidor, conforme mostra a Figura 4. A camada de desenvolvimento é composta pelos componentes Microsoft Visual Studio 2005 juntamente com .NET *Compact Framework*, que permitem a criação de aplicações para SQL Server Mobile, sendo possível criar aplicações com Microsoft Visual Basic, Microsoft Visual C# ou Microsoft Visual C++ para dispositivos móveis [21].

A camada cliente é composta pelos dispositivos que possuem as aplicações desenvolvidas para acessar o SQL Server Mobile. Quando os dispositivos não podem se conectar através de redes sem fio, eles podem se conectar ao SQL Server Mobile através do Microsoft ActiveSync.

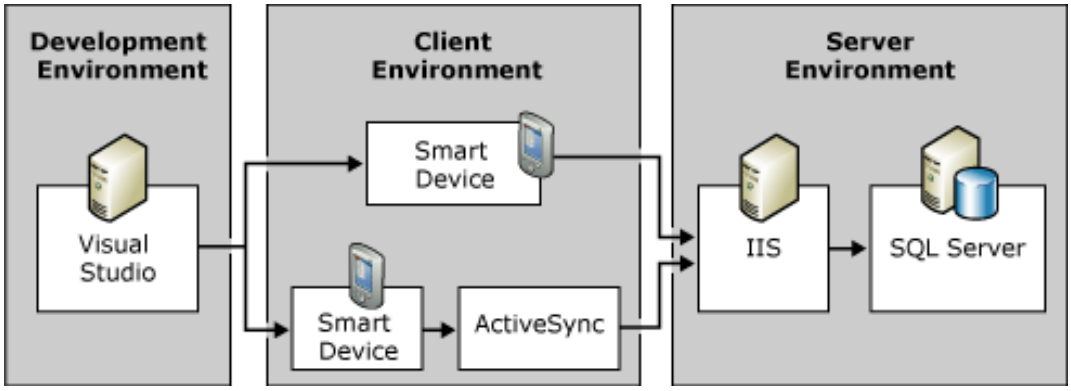

Figura 4 - Arquitetura do SQL Server Mobile [21]

A camada do servidor é composta por um ambiente que executa Microsoft Internet Information Services (IIS) e Microsoft SQL Server. O IIS é responsável pela conexão entre os dispositivos móveis e o servidor. A Figura 5 ilustra o relacionamento entre os componentes do ambiente cliente e servidor.

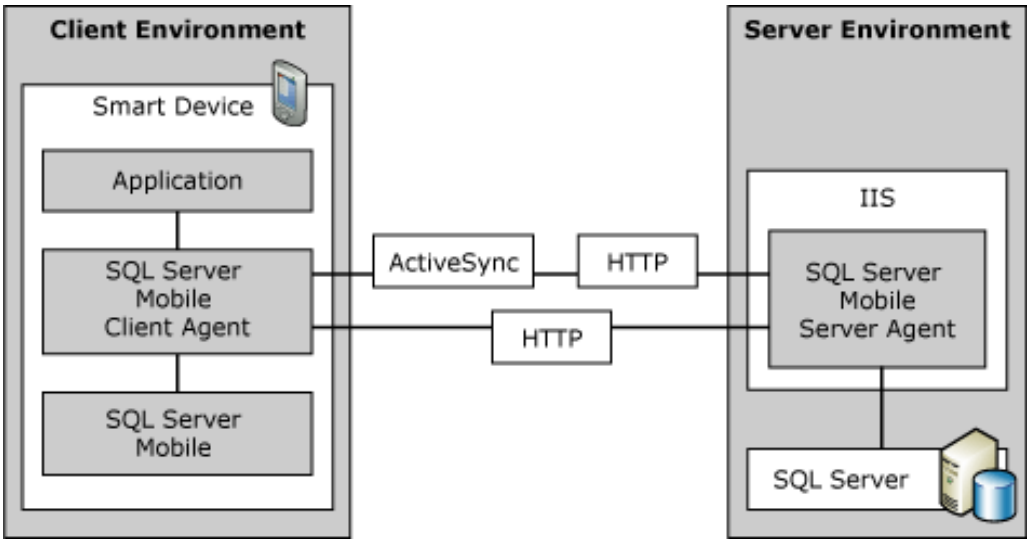

Figura 5 - Relacionamento entre os componentes do ambiente cliente e servidor [21]

A camada cliente possui a aplicação que será utilizada pelo usuário para acessar os dados e o agente cliente do SQL Mobile, sendo responsável pela configuração, conectividade e acesso aos dados no servidor [21]. Para realizar estas tarefas, este componente possui um processador de consultas, um mecanismo de armazenamento dos dados e as APIs utilizadas pelas aplicações para acessar os dados do SQL Server Mobile.

A camada existente no servidor possui o IIS, o agente servidor do SQL Mobile e o Microsoft SQL Server [21]. O componente IIS permite a integração com servidores Web, gerência de *sites* FTP (*File Transfer Protocol*) e o uso de NNTP (*Network News Transport*

Protocol) e SMTP (*Simple Mail Transfer Protocol*) para rastrear notícias e e-mails. O IIS permite que os dispositivos possam transferir dados através da RDA (*Remote Data Access*) ou replicações viabilizando o acesso seguro com padrões de Internet aos bancos de dados SQL Server Mobile. O agente servidor do SQL Mobile recebe as requisições sobre HTTP (*HyperText Transfer Protocol*) do agente cliente, conecta-se ao servidor SQL e retorna os dados e informações de esquema solicitados pelo cliente através do protocolo HTTP. O ambiente servidor ainda possui o SQL Server que é composto por um conjunto de componentes que permitem processar os dados necessários de acordo com as solicitações dos clientes. Este componente reside no servidor *desktop* e realiza o gerenciamento relacional dos dados, incluindo características de *Data Warehousing* e componentes inteligentes de negócios.

O SQL Server Mobile permite o acesso simultâneo de vários usuários suportando várias requisições a partir de múltiplas aplicações. Conforme descrito em [21], é possível ter várias aplicações acessando e modificando diferentes conjuntos de dados. Os acessos aos bancos de dados do SQL Server Mobile podem acontecer nos seguintes cenários:

- Conexão simples: uma aplicação requisita uma conexão ao banco de dados SQL Server Mobile;
- Múltiplas conexões: uma aplicação realiza várias conexões para acessar um banco de dados executando várias operações;
- Múltiplas aplicações: várias aplicações de um dispositivo podem acessar um único banco de dados ao mesmo tempo.

De acordo com [21], o SQL Server Mobile suporta trabalhar com transações garantindo as propriedades de atomicidade, consistência, isolamento e durabilidade (ACID), porém, no ambiente móvel, esta ferramenta não suporta transações distribuídas, nem trabalha com *save points*, ou seja, não permite que seja realizado o *rollback* em parte da transação. A aplicação sempre executa o *rollback* ou *commit* na transação como um todo. Esta ferramenta também permite trabalhar com bloqueios, de forma que quando uma transação é iniciada através de comandos DML ou DDL (*Data Definition Language*), o SQL Server Mobile bloqueia os recursos necessários para manter o nível de isolamento da transação em andamento. A seguir, são apresentados os níveis de granularidades de bloqueios que podem ser realizados pelo SQL Server Mobile:

- Registros;
- 4 KB de dados ou paginas de índices;
- Tabelas de esquemas de dados;
- Tabelas de dados;
- Bancos de dados.

O SQL Server Mobile disponibiliza uma aplicação cliente para acessar os dados e esquemas de bancos de dados. Uma versão do *Query Analyser* foi desenvolvida especialmente para o SQL Server Mobile para ser executada em dispositivos móveis, conforme mostra as telas apresentadas na Figura 6.

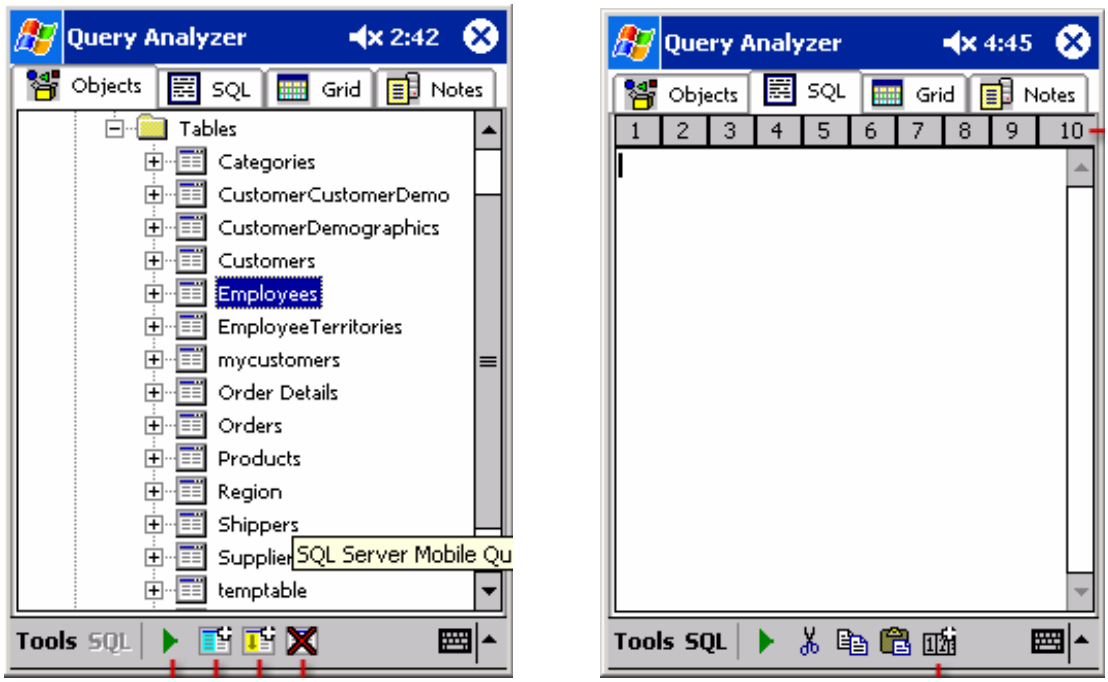

Figura 6 - Interfaces do SQL Server Mobile em um dispositivo móvel [21]

Através do *Query Analyser* é possível realizar tarefas como:

- Criação de bancos de dados;
- Conexão com outros bancos de dados do dispositivo;
- Criação e modificação de tabelas e índices no banco de dados;
- Definição de consultas através de sentenças SQL;
- Inserção e exclusão de linhas nas tabelas;

- Compactação de banco de dados.

3.2.1.1 Principais Características e Limitações

Os bancos de dados do SQL Server Mobile são armazenados em arquivos com extensões do tipo .SDF que armazenam o conteúdo completo do banco de dados e podem chegar a 4GB de tamanho. O SQL Server Mobile pode acessar e modificar os dados de tais arquivos tanto no ambiente cliente quanto no servidor. Conforme consta em [21], seguem algumas funcionalidades disponibilizadas pelo SQL Server Mobile:

- Criação, edição e exclusão de tabelas e dados;
- Criação, manipulação e exclusão de índices;
- Consultas a informações de esquema e tipos de dados;
- Utilização de comandos das linguagens DML e DDL;
- Integridade referencial e operações em cascata;
- Suporte para transações;
- Suporte para replicações de dados;
- Utilização de cursor;
- Suporte ao acesso remoto de dados (RDA – *Remote Data Access*);
- Suporte a criptografia e senha de proteção para os bancos de dados.

A Tabela 1 apresenta algumas limitações existentes no SQL Server Mobile. Muitos desses limites podem ser ainda mais restritos devido às configurações físicas do dispositivo (memória, processador) e ao sistema operacional utilizado.

Característica	**Limitação**
Nome da coluna	1028 caracteres
Quantidade de colunas por tabela	1024
Tamanho do registro	8060 *bytes*
Tamanho do banco de dados	4 GB
Nome da tabela	128 caracteres
Tamanho da tabela	512 MB
Quantidade de caracteres em sentenças SQL	Ilimitada

Característica	Limitação
Quantidade de colunas no cursor	1024
Colunas em cláusulas `Order by, Group by` ou `Distinct`	Ilimitada
Quantidade de operações por consulta	Ilimitada
Quantidade de tabelas em junções	Ilimitada
Quantidade de colunas no índice	16
Índices por tabela	249
Chave primária e chave estrangeira	Suporta
Quantidade de restrições por tabela	249

Tabela 1 – Características e limitações do SQL Server Mobile

Dessa forma, o SQL Server Mobile disponibiliza várias funcionalidades para que seja possível gerenciar os dados através do dispositivo móvel realizando suporte a múltiplos acessos, integração com SQL Server e Visual Studio 2005 e disponibilizando um conjunto de sentenças SQL para serem executadas através de um processador de consultas.

3.2.2 Sybase SQL Anywhere

O SQL Anywhere da Sybase é uma ferramenta que oferece o gerenciamento e a sincronização dos dados para ambientes computacionais móveis. O Sybase UltraLite é a tecnologia existente no SQL Anywhere Studio que reside localmente nos dispositivos móveis com o objetivo de permitir a manipulação e a distribuição de dados [44]. O SQL Anywhere tem como seus principais componentes os que seguem: *Adaptive Server Anywhere*, *UltraLite*, *MobiLink*, SQL *Remote* e ferramentas de gerenciamento de dados e desenvolvimento de aplicações [39].

3.2.2.1 Adaptive Server Anywhere

O *Adaptive Server Anywhere* é um ambiente que foi projetado para dar suporte aos sistemas de bancos de dados com características de mobilidade. Este componente é composto por várias ferramentas que possibilitam o gerenciamento e o armazenamento dos dados, o envio de sentenças SQL entre as unidades e a inclusão do servidor de banco de dados na rede [38]. Este componente foi projetado para realizar tarefas que requerem todas as características de um banco de dados relacional com condições para trabalhar tanto em

ambientes com amplos recursos quanto em ambientes com limitações de memória, processamento de CPU e área de armazenamento.

O *Adaptive Server Anywhere* possui duas versões de servidores de banco de dados: pessoal e de rede. O servidor de banco de dados pessoal é voltado para disponibilizar acesso local para um usuário sem acesso à rede, já o servidor de rede permite trabalhar com a plataforma cliente-servidor disponibilizando acesso remoto aos clientes através da rede [38]. A única diferença entre as duas versões é o suporte a comunicação.

3.2.2.2 UltraLite

UltraLite é um sistema de banco de dados relacional para dispositivos móveis que provê gerenciamento de dados e possibilidade de sincronização destes dados com um banco de dados centralizado [43]. Cada banco de dados do tipo *UltraLite* permite ser acessado apenas por uma aplicação. Caso seja necessário que mais de uma aplicação acesse o mesmo banco de dados, é recomendado utilizar o *Adaptive Server Anywhere*, uma vez que o *UltraLite* é mais indicado para os dispositivos portáteis de pequeno porte como os PDAs.

Esta aplicação disponibiliza várias *interfaces* de programação e integração com ferramentas de programação como Visual Studio .NET, AppForge MobileVB e Crossfire, Borland JBuilder e eMbedded Visual Basic. Para tanto ela disponibiliza vários componentes que possibilitam desde o simples acesso aos dados até a execução de consultas SQL mais complexas. Conforme descrito em [43], seguem alguns destes componentes:

- UltraLite para MobileVB: para desenvolvimento em Microsoft Visual Basic;
- UltraLite ActiveX: para desenvolvimento usando Mbedded Visual Basic ou JScript com Pocket IE;
- Native UltraLite: para Java;
- UltraLite.NET: para desenvolvimento usando Visual Studio .NET;
- Componente UltraLite C++: para desenvolvimento usando C++;
- UltraLite M-Business Anywhere;
- ODBC.

UltraLite também disponibiliza *interfaces* estáticas para o desenvolvimento utilizando C++ e SQL embutido ou Java utilizando *interface* JDBC.

O *UltraLite* provê o armazenamento, a recuperação e a manipulação dos dados através de funções SQL, de forma que seu interpretador permite trabalhar com tabelas, tipos de dados, índices, restrições de chave e restrições de integridade, processamento de transações, cursores, operações de junção entre tabelas e segurança de dados através de criptografia [43]. O *UltraLite* permite criar banco de dados com tabelas, colunas e índices que resultam em uma melhor performance quando se tratam de dispositivos móveis. Também permite acessar e enviar informações para bancos de dados coorporativos através de processos de sincronização.

3.2.2.3 SQL Remote

SQL *Remote* é uma tecnologia de sincronização baseada em mensagem e consiste de um banco de dados central que armazena uma cópia mestre e dos bancos de dados remotos existentes em dispositivos móveis [42]. A entrada dos dados pode ocorrer no banco de dados central ou nas unidades móveis. A sincronização acontece baseada em um sistema de mensagem, como e-mail ou transferência de arquivo, conforme ilustra a Figura 7.

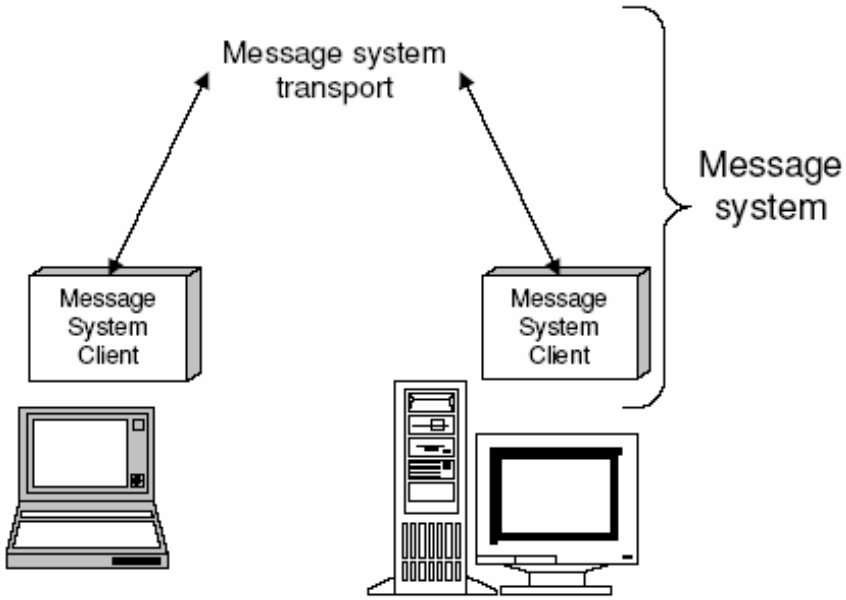

Figura 7 - Sistema de mensagens utilizado pelo SQL Remote [39]

Existe um agente de mensagem em cada banco de dados que envia uma mensagem de replicação para o banco de dados central a cada alteração nos dados. Este

mesmo agente é responsável por receber mensagens dos demais bancos de dados e modificar o banco de dados local de acordo com o conteúdo existem na mensagem recebida. Este componente implementa um protocolo que garante a ordem correta das replicações a serem realizadas.

3.2.2.4 MobiLink

O *Mobilink* tem uma forma de sincronização de tempo real baseada em sessão que requer uma conexão direta com o banco de dados consolidado ou com o *Adaptative Anywhere Server* [41]. A conexão pode acontecer através de protocolos TCP/IP ou por sincronização *HotSync* utilizando conexão por porta serial ou ondas de rádio [39]. O *MobiLink* é indicado para cenários que possuem conexão com o banco de dados *UltraLite* ou com um banco de dados central com *Adaptative Anywhere Server*.

A Tabela 2 apresenta um comparativos entre algumas características do SQL *Remote* e o *MobiLink*.

Características	***SQL Remote***	***MobiLink***
Método de sincronização	Baseado em mensagem	Baseado em sessão
Sincronização bidirecional	Sim	Sim
Sybase ASA ou ASE como banco de dados central	Sim	Sim
Suporte a Oracle, Microsoft, IBM	Não	Sim
Sybase ASA como banco de dados remoto	Sim	Sim
Sybase Ultralite como banco de dados remoto	Não	Sim
Opera em modo desconectado	Sim	Não

Tabela 2 - Tabela comparativa entre SQL *Remote* e *MobiLink* [42]

Conforme [39], o *MobiLink* inicia uma conexão com o servidor de sincronização e carrega a lista completa com todas as alterações realizadas no banco de dados remoto desde a última sincronização. Após receber os dados, o servidor atualiza o banco de dados central e, em seguida, retorna para o banco de dados remoto as suas alterações. O banco de dados remoto recebe este conjunto de alterações, envia uma confirmação de recebimento ao banco de dados central e finaliza a conexão.

Assim, o SQL Anywhere disponibiliza através de seus componentes um conjunto de funcionalidades que permitem manter um banco de dados em dispositivos móveis. Esta ferramenta permite que o banco de dados central e os bancos de dados remotos possam sempre ser atualizados em funções de alterações realizadas em um dos bancos de dados.

3.2.2.5 Principais Características e Limitações

Todas as tabelas de um banco de dados necessitam de uma chave primária definida, pois elas são necessárias durante o processo de sincronização dos dados do dispositivo móvel para o banco de dados central. As seguintes sentenças SQL podem ser utilizadas em aplicações que acessam bancos de dados *UltraLite*:

- `Select`, `Insert`, `Update` e `Delete`;
- `Truncate Table`;
- `Commit` e `Rollback`;
- `Start`/`Stop Synchronization Delete`.

Por outro lado, de acordo com [43], o *UltraLite* não suporta as seguintes funcionalidades:

- Utilização de sentenças de `Delete` e `Update` em cascata;
- Colunas computadas;
- Modificação de esquema: para alterar o esquema do banco de dados é necessário criar uma nova versão da aplicação;
- Criação de tabelas temporárias;
- Utilização de *Stored Procedures* e *Triggers*;
- Acesso a tabelas e funções de sistema.

A Tabela 3 mostra algumas das principais limitações do *UltraLite*. Muitos desses limites podem ser ainda mais restritos devido às configurações do dispositivo (memória, processador) e ao sistema operacional utilizado.

Aspectos	Limitações
Número de coluna por tabela	65535 (limitado pelo tamanho da linha - aproximadamente 4KB)

Aspectos	Limitações
Número de índices	Aproximadamente 1000
Número de linhas por banco de dados	Limitado a memória disponível
Número de linhas por tabela	65533
Número de tabelas por banco de dados	Aproximadamente 1000
Tamanho da linha	Aproximadamente 4 KB
Tamanho máximo do banco de dados	2 GB
Número de conexões por banco de dados	14
Número de tabelas referenciadas por transação	Ilimitado

Tabela 3 - Principais limitações do Sybase UltraLite [43]

3.2.3 Oracle Database Lite

O Oracle Lite é uma ferramenta da Oracle que gerencia bancos de dados existentes em dispositivos móveis como PDAs *(laptops* e *handhelds)*. Pode ser executado nas plataformas Windows 98/NT/2000/XP, Windows CE/Pocket PC, Palm OS e Embedded Linux e permite trabalhar com JDBC, ODBC, ADO.NET e SODA (*Simple Object Data Access*) [23].

Um dos objetivos da ferramenta é permitir a integração de estruturas de bancos de dados com a Internet e proporcionar o acesso aos dados independentemente da plataforma utilizada. Algumas características do Oracle Database Lite são:

- Suporte ao SQL 92 e propriedades ACID;
- Suporte a *Stored Procedures* e *Triggers*;
- Suporte a múltiplas conexões simultaneamente;
- Bibliotecas compartilhadas da Palm.

Conforme mostra a Figura 8, o Oracle Database Lite é formado por componentes que pertencem a três camadas distintas: a camada cliente, a camada intermediária e a camada do servidor. Estas camadas possuem os seguintes componentes:

- *Oracle Lite Database*: sistema de banco de dados relacional projetado especificamente para dispositivos móveis como PDAs (*laptops* e *handhelds*);

- *Mobile Server*: realiza o processo de sincronização dos dados entre os dispositivos móveis e o servidor de banco de dados central;
- *Life Cycle Management*: processo responsável pela distribuição, instalação e gerenciamento das aplicações, dados e arquivos no dispositivo móvel. Também é responsável pelo gerenciamento de usuários, dispositivos e sistema;
- *Rapid Application Development*: ambiente que possibilita o desenvolvimento de aplicações para as plataformas Microsoft Windows CE, Pocket PC, Windows NT/2000/XP/2003, Linux e Palm OS.

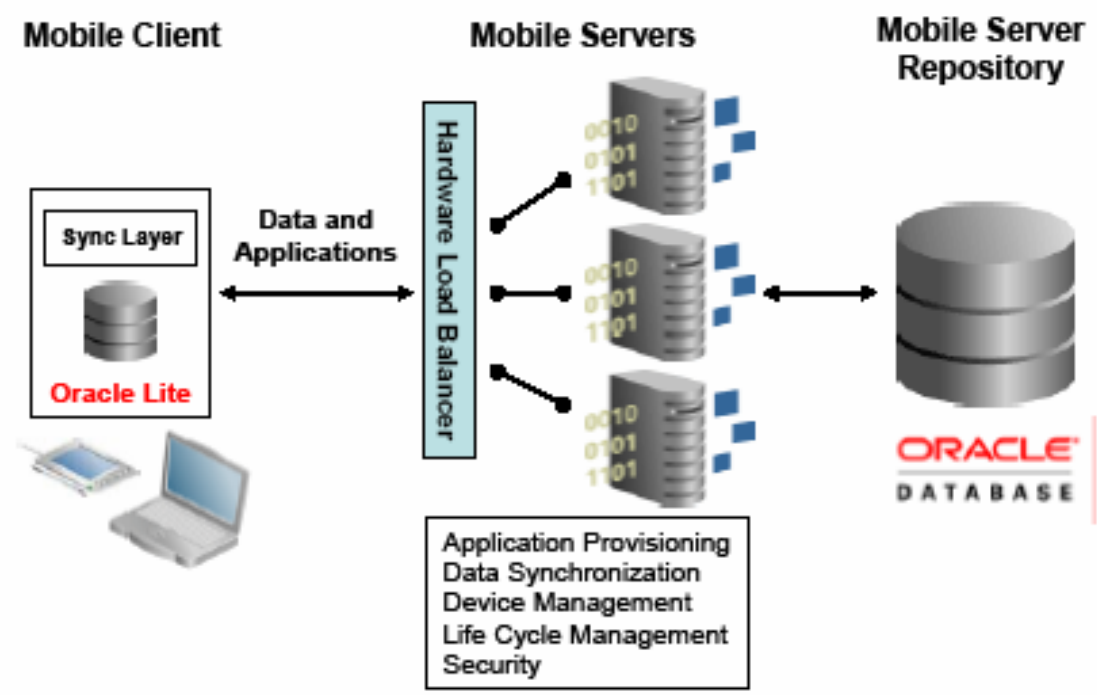

Figura 8 - Visão Geral dos componentes do Oracle Lite [23]

A Figura 9 apresenta a arquitetura do Oracle Database Lite com os componentes existentes em cada camada. Quando o sistema é instalado em produção, a camada cliente possui apenas os componentes *Mobile* SQL, *Mobile Sync* e o RDBMS. A camada intermediária possui os componentes *Mobile Server* e o *Message Generator and Processor* (MGP). A camada do servidor possui o servidor de banco de dados Oracle atuando como o *Mobile Server Repository*. Tais componentes estão divididos nestas camadas formando dois grupos: *Development Support* e *Mobile Server*.

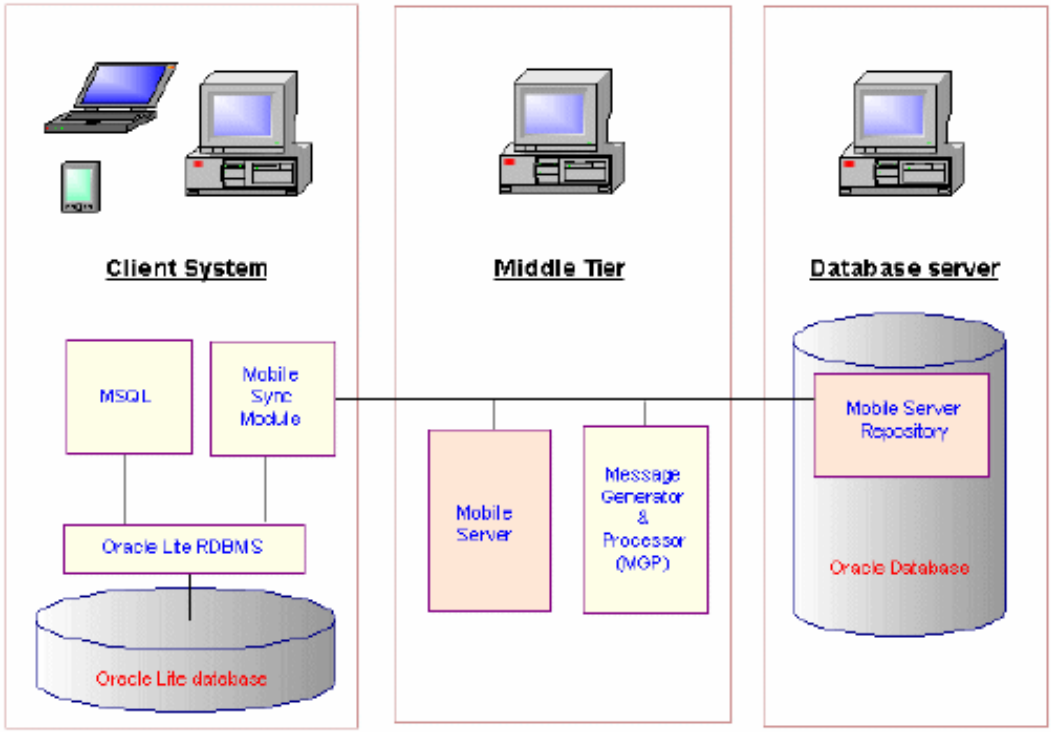

Figura 9 - Arquitetura do Oracle Database Lite [26]

3.2.3.1 Development Support

O *Development Support* contém os componentes que oferecem a infra-estrutura necessária para o desenvolvimento e instalação de aplicações móveis. Permite trabalhar com JDBC, ODBC, ADOCE, ADO.Net e com linguagens de programação como Java, C/C++ e Visual Basic. A instalação do *Mobile Development Kit* (MDK) inclui o Oracle Database Lite [22].

Para desenvolver aplicações usando Oracle Lite é necessário instalar o *Mobile Development Kit* que possui os seguintes componentes:

- *Oracle Lite* RDBMS: sistema de gerenciamento de banco de dados relacional;
- *Mobile Sync*: responsável pela sincronização dos dados e reside na unidade móvel;
- *Mobile* SQL: permite manipular os dados a partir de um dispositivo móvel;
- *Packaging Wizard*: ferramenta que permite publicar as aplicações no *Mobile Server*;
- *Mobile Development Workbench*: ferramenta que permite o desenvolvimento de aplicações para acessar Oracle Lite.

Para que uma aplicação possa ser executada em um dispositivo móvel, ela deve ser publicada no *Mobile Server* através do *Packaging Wizard*. A instalação da aplicação é feita instalando o Oracle Lite no dispositivo e, em seguida, deve ser executado o *Mobile Sync* a partir do dispositivo. O *Mobile Sync* acessa o *Mobile Server* e atualiza todas as aplicações e dados necessários.

O *Oracle Lite* RDMS (*Relational Database Management System*) é um gerenciador de banco de dados criado especificamente para PDAs (*laptops* e *handhelds*), fazendo parte da camada cliente e sendo utilizado como o SGBD local quando o usuário está desconectado do servidor de banco de dados central, conforme mostra a Figura 9.

O *Mobile* SQL (MSQL) é uma ferramenta que permite criar, acessar e manipular os bancos de dados, tabelas e visões através de sentenças SQL definidas a partir de um dispositivo móvel. O MSQL é instalado com o *Mobile Development Kit* e faz parte da camada cliente, conforme ilustra a Figura 9.

De acordo com [26], o *Mobile Sync* é instalado na unidade móvel e permite sincronizar os dados dos dispositivos móveis com os dados do servidor (*Mobile Server Repository*) compondo também a camada cliente. Quando é executado, o *Mobile Sync*, inicialmente, obtém informações do usuário para autenticá-las junto ao *Mobile Server*. Após a autenticação, são coletados os dados alterados para serem transmitidos para o *Mobile Server* para, em seguida, aplicar as alterações encontradas nos bancos de dados *Oracle Lite*. Ele permite que os dados possam ser sincronizados entre o servidor de dados coorporativo e os bancos de dados Oracle Lite existentes nos dispositivos móveis e vice-versa. As alterações são identificadas pelo *Mobile Server* para cada usuário a partir da publicação dos itens alterados. O *Mobile Sync* comunica-se com o *Mobile Server* identificando informações do usuário para autenticação junto ao *Mobile Server*. Este coleta as alterações feitas no Oracle Lite e aplica no banco de dados centralizado. Adicionalmente, o *Mobile Sync* pode realizar operações de criptografia e compressão dos dados transmitidos.

3.2.3.2 Mobile Server

O componente *Mobile Server* é formado pelo *Resource Manager* e o *Consolidator Manager*. O *Resource Manager* realiza a publicação, distribuição e instalação das aplicações. O *Consolidator Manager* atua na sincronização e a replicação dos dados e aplicações [26]. Dessa forma, o componente *Mobile Server* é responsável por dar suporte à replicação entre o

cliente e o servidor. Ele também atua como um *servlet,* pois tanto disponibiliza o suporte para a replicação do cliente como provê um ambiente para a execução de aplicações baseadas na Web. O *Mobile Server* possui um servidor de sincronização bidirecional que utiliza a tecnologia Java *Servlet* para instalar os componentes no dispositivo móvel. São tarefas executadas pelo *Mobile Server*:

- Suporte ao armazenamento e sincronização de dados;
- Gerenciamento das aplicações, dos dispositivos e dos usuários;
- Publicação de aplicações;
- Instalação e atualização de aplicações.

O *Message Generator and Processor* (MGP) é responsável por identificar algum conflito de mudanças entre os dados do cliente e do servidor, sendo executado através de ciclos. Cada ciclo contém as fazes de aplicação e composição, conforme mostra a Figura 10.

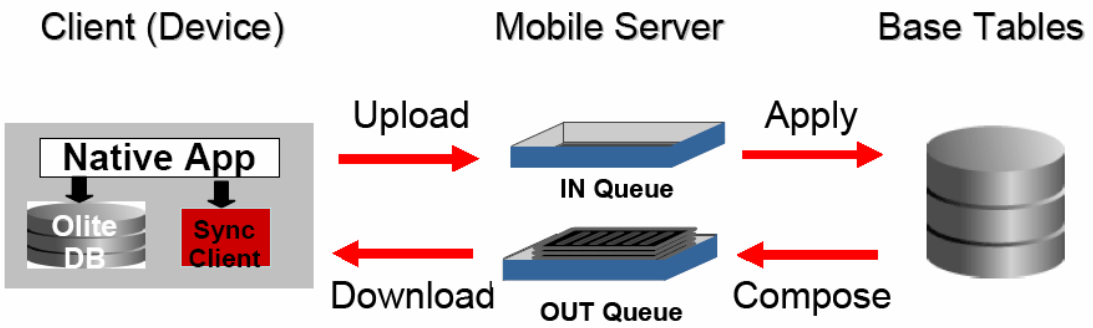

Figura 10 - Processo de sincronização do Oracle Lite [23]

De acordo com [23], na fase de aplicação o MGP identifica os dados alterados no cliente e aplica no servidor de banco de dados através de uma transação única. Após a fase de aplicação, é iniciada a fase de composição, que prepara todas as alterações realizadas no servidor que devem ser atualizadas no cliente. Dessa forma, o MGP participa do processo de sincronização gerenciando as filas de entrada e saída das alterações realizadas tanto nos bancos de dados clientes, quanto no central. O processo de sincronização segue os seguintes passos: (1) *Mobile Sync* move as alterações feitas no cliente para fila de entrada (*In Queue*) e da fila de saída (*Out Queue*) para os clientes; (2) MGP aplica as alterações da fila de entrada nas tabelas do banco de dados centralizado e compõe as alterações do banco de dados central para a fila de saída que, em seguida, será atualizado nos bancos de dados dos dispositivos clientes.

O *Mobile Server Repository* contém todas as informações necessárias para executar o *Mobile Server* [26]. Tais informações são, geralmente, armazenadas no mesmo banco de dados em que são armazenados os dados coorporativos.

3.2.3.3 Principais Características e Limitações

Oracle Database Lite possui compatibilidade com SQL 92, possui as propriedades ACID, permite trabalhar com *stored procedures* e *triggers* baseadas em Java, possui *interface* com JDBC nativo, ODBC habilitado para todas as plataformas e bibliotecas compartilhadas para o ambiente Palm OS.

Conforme descrito em [25], outras características encontradas no Oracle Lite são:

- Suporte a tabelas, visões e índices;
- Suporte a segurança (usuários e regras);
- Suporte ao SQLJ e SQL padrão;
- Suporte a *stored procedures* e *triggers*;
- Suporte à sincronização em redes *wireless*;
- Acesso a classes Java;
- Suporte a múltiplas *interfaces*: ODBC, JDBC, SQLJ, OKAPI, JAC;
- Possibilidade de utilização de múltiplas ferramentas de desenvolvimento como: Visual C++, Visual Coffee, Oracle JDeveloper e Developer, Delphi, SatelLite Forms, CodeWarrior;
- Suporte a vários sistemas operacionais como: Windows CE, Windows 95/98/NT, Palm OS, Symbian EPOC;
- Compatibilidade com o Oracle Database Servers*;*
- Suporte para serviço de voz;
- Suporte para criptografia para a segurança dos dados.

A Tabela 4 mostra o resumo dos principais requisitos e limitações do Oracle Database Lite. Muitos desses limites podem ser ainda mais restritos devido às configurações do dispositivo (memória, processador) e ao sistema operacional utilizado.

Aspectos	Requisitos/Limitações
Tamanho do banco de dados	4 GB
Plataformas suportadas	Palm OS, Pocket PC, Win32 e Linux
Requisitos de *hardware*	1 MB RAM, 5 MB memória de armazenamento
Tamanho do nome do banco de dados	> 8 caracteres
Tamanho do nome da tabela	30 caracteres
Tamanho do nome da coluna	30 caracteres
Tamanho do nome da visão	30 caracteres
Número máximo de colunas para índice	32
Número máximo de colunas por tabela	1000

Tabela 4 - Principais limitações e requisitos do Oracle Lite [24]

3.2.4 DB2 Everyplace Database

O DB2 Everyplace (DB2E) é um banco de dados relacional da IBM que permite acessar dados de um servidor de banco de dados central a partir de dispositivos móveis, realizando sincronização bidirecional de arquivos entre a origem de dados corporativa e os dispositivos remotos [12]. Para tanto, o DB2 Everyplace possui os seguintes componentes:

- Mecanismo de banco de dados DB2 Everyplace: *software* que é executado no dispositivo remoto e permite que os usuários acessem e modifiquem a cópia local dos dados.
- *DB2 Everyplace Sync Server*: um programa cliente/servidor que gerencia a sincronização bidirecional de dados entre o banco de dados local e o remoto.

3.2.4.1 O Banco de Dados Móvel do DB2 Everyplace

O banco de dados DB2 Everyplace reside no dispositivo móvel e está disponível para as plataformas Palm OS, Symbian OS, Windows CE/Pocket PC, Linux incorporado em dispositivos do tipo *Sharp Zaurus* [13].

3.2.4.2 DB2 Everyplace Sync Server

O *DB2 Everyplace Sync Server* é responsável pela sincronização dos dados entre as unidades móveis e o servidor central, atuando como um intermediário entre o *software*

cliente de sincronização no dispositivo remoto e o banco de dados DB2 UDB ou JDBC no servidor de origem [13].

A Figura 11 ilustra um cenário de desenvolvimento de aplicativos e de sincronização de dados entre um dispositivo móvel e um servidor central. O ambiente de desenvolvimento possui o banco de dados DB2 Everyplace que é executado no dispositivo móvel e que permite que os usuários acessem e modifiquem uma cópia local dos dados; e o *Mobile Application Builder* que disponibiliza as aplicações que podem acessar localmente os dados dos dispositivos. O ambiente de sincronização mostra os componentes que possibilitam a comunicação de dados entre o dispositivo móvel e a base central. Para iniciar um processo de sincronização, o componente cliente *DB2E Sync Client*, existente no dispositivo móvel, realiza a negociação para se comunicar com o servidor *DB2 Everyplace Sync Server* através do *Mid-tier* que, por sua vez, realiza a comunicação com a fonte de dados centralizada.

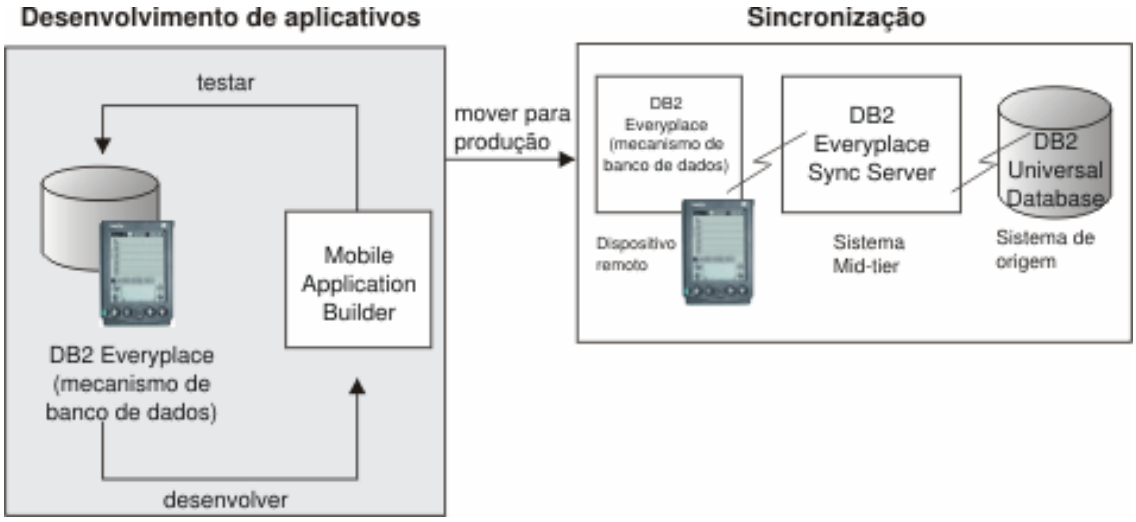

Figura 11 - Cenários de desenvolvimento e sincronização do DB2 Everyplace [13]

Para realizar a sincronização dos dados, o *Sync Server* submete as mudanças que foram feitas em cópias locais pelos usuários móveis para o servidor central. Em seguida, os usuários móveis recebem as mudanças que foram feitas no banco de dados servidor e que não estão na cópia local do dispositivo móvel, realizando a sincronização do servidor para o cliente. Assim, o *Sync Server* é um servidor de sincronização bidirecional que possui a finalidade de mover dados entre o DB2 Everyplace Database dos dispositivos móveis e o banco de dados coorporativo. [12]

O *Mobile Devices Administration Center* é uma ferramenta de administração gráfica do *Sync Server* que permite definir subconjuntos de dados e arquivos a serem acessados por grupos de usuários. O *Sync Server* acessa estas informações de administração cada vez que um usuário pede uma sincronização dos dados. Ao realizar a sincronização, o *DB2 Everyplace Sync Server* é capaz de resolver conflitos e realizar a compressão e criptografia dos dados para uma transferência mais segura e rápida. A ferramenta de administração fornecida pelo *Sync Server* ajuda a gerenciar e fornecer serviços de sincronização para grupos de usuários com necessidades semelhantes.

De acordo com [12], o *DB2 Everyplace Sync Server* suporta a sincronização de bancos de dados dos seguintes tipos:

- DB2 Universal Database V8.1 e V8.2 para Windows, Linux, AIX, HP-UX e Solaris;
- DB2 Universal Database V7.1 e V8.1 para z/OS;
- DB2 Universal Database V5R1 e V5R2 para IBM iSeries;
- Informix Dynamic Server 9.4 (com Informix JDBC);
- IBM Cloudscape V10;
- Lotus Domino Server 6.0.2;
- Oracle 9i e 10g;
- Microsoft SQL Server 2000 com Service Pack 3a (V8.00.760) (com SQL Server Driver para JDBC);
- Sybase Adaptive Server Enterprise 11.93 (com Sybase J-Connect 5.5 para JDBC).

3.2.4.3 Everyplace Sync Client

O *Everyplace Sync Client*, executado em dispositivos móveis, é uma API que as aplicações utilizam para possibilitar a sincronização com o *Everyplace Sync Server* [13]. O *Everyplace Sync Client* manipula a sincronização bidirecional de dados relacionais corporativos com o banco de dados remoto do DB2 Everyplace existente no dispositivo móvel e gerencia as operações que serão realizadas pelos usuários localmente.

De acordo com [13], o *Sync Client* está disponível para as seguintes plataformas:

- Palm OS 3.5, 4.1 e 5.0 ou versões superiores;
- Windows CE 3.0, PocketPC, PocketPC2002, HandheldPC, Windows CE.Net, e Windows Mobile 2003 para PocketPC;
- Symbian V6 e V7;
- Neutrino 6.2;
- Linux;
- Windows NT/2000/XP;
- Java 2 Micro Edition Mobile Information Device Profile virtual machine.

3.2.4.4 Principais Características e Limitações

A seguir são descritas algumas características do DB2 Everyplace [13]:

- Tamanho reduzido (200 Kb);
- Possui APIs de banco de dados padrões;
- Possui *Query-By-Example* (QBE) como *interface* de consulta;
- Suporte a transações;
- Possui suporte a SQL básico, funções agregadas, operações relacionais, como `Join, Group by` e `Order by`;
- Oferece suporte as linguagens C/C++ e Java.
- Suporte a criptografia dos dados.

A Tabela 5 mostra o resumo das principais limitações do DB2 Everyplace Database. Muitos desses limites podem ser ainda mais restritos devido às configurações do dispositivo (memória, processador) e ao sistema operacional utilizado.

Aspectos	Limitações
Tamanho máximo da tabela	2 GB
Número máximo de tabelas em um banco de dados	65535
Número máximo de índices em uma tabela	15
Comprimento máximo de uma linha em uma tabela	64 KB

Aspectos	Limitações
Número máximo de chaves estrangeiras em uma tabela	8
Número máximo de colunas em um índice	8
Número máximo de colunas em uma chave primária	8
Número máximo de conexões para um caminho de banco de dados	1
Número máximo de linhas em uma tabela	Limitada pelo tamanho da tabela
Número máximo de colunas em uma tabela	256
Comprimento máximo para uma coluna `CHAR`	32767 *bytes*
Comprimento máximo para coluna `VARCHAR`	32767 *bytes*
Comprimento máximo cumulativo para colunas de comprimento fixo de 32767 linhas	32767 *bytes*
Tamanho máximo de decimais	31 dígitos
Comprimento máximo de cada coluna em um único índice	1024 *bytes*

Tabela 5 - Principais limitações do DB2 Everyplace [13]

3.3 Análise Comparativa entre os Bancos de Dados Móveis

Os sistemas gerenciadores de banco de dados móveis (SGBDs Móveis) descritos neste capítulo apresentam várias particularidades e características que visam contemplar as necessidades existentes nos ambientes com suporte a mobilidade como segurança, sincronização, acesso remoto, consistência dos dados e compactação dos dados.

Apesar de várias outras características serem encontrados nas ferramentas apresentadas, a Tabela 6 mostra uma análise comparativa que considera apenas os aspectos relacionados à arquitetura de armazenamento dos dados por ser este o foco deste trabalho.

Aspectos	Microsoft SQL Mobile	Sybase SQL Anywhere	Oracle Database Lite	DB2 Everyplace Database
Restrições de Integridade	Sim	Sim	Sim	Sim
Modificação de Esquema	Sim	Não	Sim	Sim
Compactação do Banco de Dados	Sim	Sim	Não	Sim

Tabela 6 - Comparativo entre as ferramentas de bancos de dados móveis

Todas as ferramentas apresentam as características necessárias para definir e manter as restrições de integridade em seus bancos de dados. A possibilidade de modificar o esquema de dados não foi encontrada na ferramenta da Sybase e técnicas de compactação dos dados não são disponibilizadas na ferramenta da Oracle. Apesar de existir no mercado ferramentas robustas e com várias funcionalidades que atendem as necessidades existentes em ambientes móveis, percebe-se que alguns aspectos importantes para dispositivos móveis de pequeno porte, como a aplicação de técnicas de compactação dos dados, não são encontradas em todas estas ferramentas. Com o objetivo de apresentar uma proposta que disponibilize as características essenciais de um SGBD que realize o gerenciamento de dados em dispositivos móveis com recursos computacionais limitados como PDAs, no Capítulo 4 é apresentada uma estratégia para o gerenciamento e armazenamento de dados em tais equipamentos.

Capítulo 4

Uma Estratégia Eficiente para Armazenamento e Acesso a Dados em Dispositivos PDAs

Este capítulo apresenta a proposta deste trabalho, que consiste na definição de uma estratégia de armazenamento e acesso a dados em dispositivos com recursos computacionais limitados.

4.1 Introdução

Este capítulo descreve uma estratégia de armazenamento eficiente dos dados em dispositivos móveis do tipo PDA. A estrutura de armazenamento proposta disponibiliza uma visão relacional dos dados e apresenta características básicas de um sistema gerenciador de banco de dados convencional como a facilidade na evolução de esquemas de dados e a utilização de restrições de integridade. Nesta proposta, os métodos de armazenamento e compressão utilizados levam em conta as particularidades destes dispositivos.

A proposta apresentada neste trabalho parte do princípio de que para se encontrar em dispositivos móveis do tipo PDA propriedades básicas de um SGBD que ofereçam um ambiente consistente dos dados, faz-se necessário adquirir ferramentas de mercado, como as descritas no Capítulo 3. Conforme já mencionado, tais aplicativos nem sempre oferecem todas as propriedades desejáveis de serem encontradas em dispositivos móveis com recursos computacionais limitados como os *handhelds* e *palmtops*.

A importância desta proposta está na possibilidade de não mais trabalhar com os conceitos de arquivos de dados existentes nos sistemas de arquivos convencionais destes dispositivos. A principal contribuição é permitir que nestes ambientes os dados possam ser tratados através de uma visão relacional garantindo sua consistência e disponibilizando funcionalidades que facilitem a manipulação dos dados considerando um ambiente com limitação de recursos computacionais.

Dessa forma, esta estratégia de armazenamento difere-se das demais por oferecer condições mais favoráveis para otimizar a aplicação de técnicas de compressão associada às funcionalidades que permitem tratar os dados através de um nível de abstração que facilita a manipulação dos dados e a manutenção das estruturas de armazenamento.

É importante destacar que esta abordagem pode ser implementada em qualquer sistema operacional desenvolvido para PDAs como, por exemplo, Palm OS ou Pocket PC. Contudo, para demonstrar a eficiência do mecanismo proposto, será utilizada a plataforma Palm OS como um estudo de caso para os experimentos realizados. Por esta razão, algumas características deste ambiente são ressaltadas ao longo deste capítulo com o objetivo de contextualizar devidamente todas as decisões tomadas durante o processo de definição da estratégia proposta.

Na seção 4.2 é apresentada a estratégia de armazenamento proposta, suas principais características e funcionalidades como o mecanismo de acesso aos dados e a estrutura de armazenamento dos esquemas. Na seção 4.3 é apresentado o mecanismo de gerenciamentos das estruturas definidas nesta proposta. A seção 4.4 descreve sobre a importância da compressão de dados nestes ambientes, as técnicas de compressão utilizadas na arquitetura e as vantagens que estratégia proposta oferece para a aplicação de mecanismos de compressão dos dados. A seção 4.5 apresenta os trabalhos relacionados e o diferencial da estrutura proposta em relação aos demais trabalhos existentes sobre este assunto.

4.2 Estratégia de Armazenamento de Dados

Os sistemas de arquivos de ambientes operacionais para PDAs armazenam seus dados seguindo uma estrutura semelhante à estrutura tradicional de armazenamento de dados da maioria dos sistemas de arquivos convencionais. Nestes sistemas, um registro pode ser composto por vários atributos que, por sua vez, podem ser de diferentes tipos de dados. Tomando como exemplo o ambiente Palm OS, um arquivo do tipo PDB é formado por um conjunto de registros. Um registro de dado deste tipo de arquivo é semelhante a uma tupla de uma tabela de um banco de dados tradicional. Para acessar todos os campos definidos em um determinado arquivo PDB, faz-se necessário ler todos os registros deste arquivo para acessar a posição do campo desejado, no caso `Endereço`, conforme ilustrado na Figura 12.

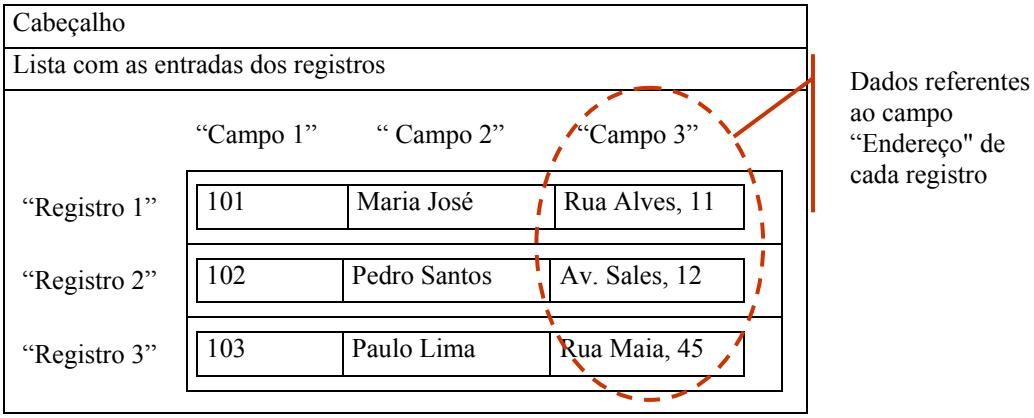

Figura 12 - Estrutura de armazenamento de um sistema de arquivos para PDAs

Essa estrutura tradicional apresenta algumas dificuldades relacionadas à manipulação de esquema como, por exemplo, a alteração na definição do registro que compõe um arquivo. Caso seja necessário incluir um novo campo, será preciso criar uma nova estrutura de arquivo e associá-la ao novo conjunto de tipos de dados que formam o registro. De forma dinâmica, não é possível incluir um novo campo a um arquivo já criado. Isso dificulta a alteração dos esquemas de dados já existentes, pois uma vez definida a estrutura, ela não pode mais ser alterada.

Da mesma forma, na estrutura tradicional de armazenamento de dados dos sistemas de bancos de dados existentes no mercado, uma tabela é formada por vários atributos em que os valores dos atributos são armazenados em disco formando uma tupla da tabela, conforme mostra a Figura 13. Esta é a estrutura tradicional de armazenamento de dados implementada pelos sistemas de bancos de dados tradicionais como Microsoft SQL Server, Oracle, Sybase e DB2.

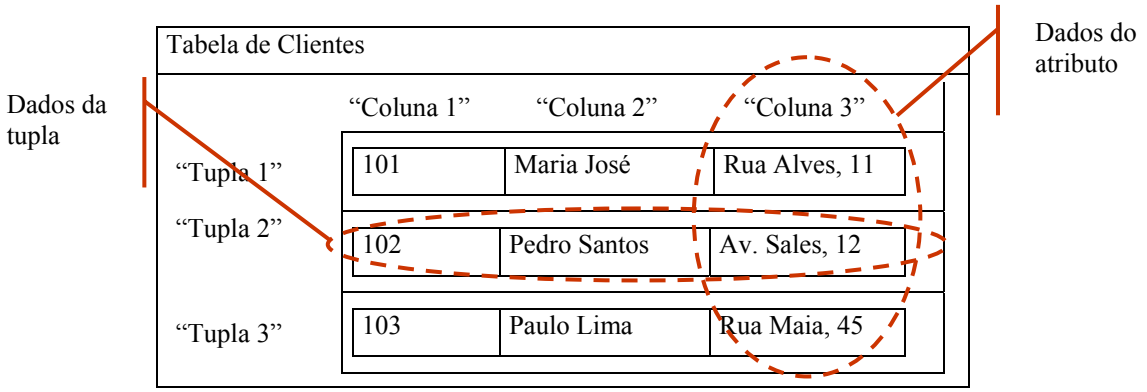

Figura 13 - Estrutura de armazenamento dos sistemas de bancos de dados tradicionais

4.2.1 Estratégia de Armazenamento de Dados Baseada em Colunas

Este trabalho apresenta um novo formato de armazenamento destes dados dentro da estrutura de arquivos de um PDA. A idéia principal é considerar que cada registro (*chunk*) armazena informações de um mesmo tipo de dado, correspondendo a um mesmo campo de dados de um registro do arquivo. Dessa forma, cada campo é armazenado separadamente em registros distintos. Os dados de um mesmo campo estão dispostos seqüencialmente no mesmo registro e são acessados através de operações de *offset.* Estas operações permitem identificar o início e o término de cada campo mais facilmente de acordo com o tipo definido, uma vez que todos os dados de um mesmo registro do arquivo são do mesmo tipo e, conseqüentemente, do mesmo tamanho. Assim, os registros que antes armazenavam os dados de todos os campos do arquivo, agora passam a armazenar apenas os dados de um único campo.

A Figura 14 ilustra a estrutura de armazenamento proposta neste trabalho, em que os dados de um mesmo campo ficam dispostos em um mesmo registro físico, seguindo a idéia de que o que antes era armazenado em uma coluna, com os mesmos campos e tipos de dados, agora passa a ser armazenado em um registro do arquivo. Para identificar o registro lógico dos dados, deve-se considerar uma associação entre os dados de mesma posição no registro.

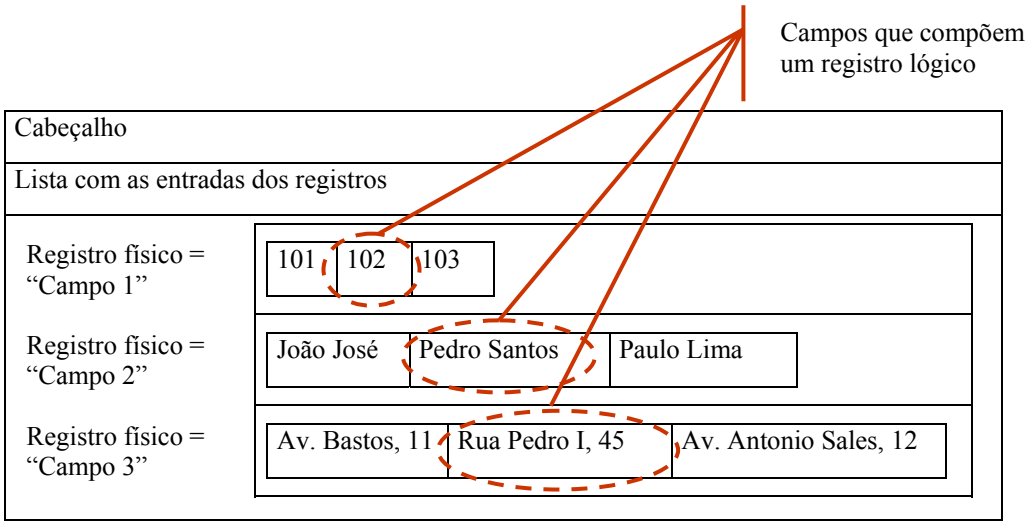

Figura 14 - Estrutura de armazenamento de dados proposta

No exemplo acima, o registro lógico é formado pelos campos que armazenam o código, o nome e o endereço do cliente. Assim, para recuperar um determinado registro na nova estrutura, deve-se identificar a posição do registro desejado e, de acordo com esta posição, recuperar todos os demais dados. Pode-se observar, pela Figura 14, que a posição física dos campos de um mesmo registro lógico não é necessariamente a mesma posição para

todos os campos, pois esta posição leva em consideração o tipo de dado definido, bem como o espaço em memória necessário para armazená-lo de acordo com seu tamanho.

A mudança principal é exatamente trocar a idéia de coluna por linha e vice-versa. Ou seja, um registro passa a armazenar os dados de um mesmo campo seqüencialmente em memória. Com essa nova estrutura, os arquivos que antes eram orientados a registros, agora passam a ser orientados a colunas, pois os dados que estão armazenados em um mesmo registro são de um mesmo tipo e estão dispostos seqüencialmente em memória, representando os dados de uma coluna, conforme o Campo 2 da Figura 14 que armazena seqüencialmente os dados do nome do cliente. Considerando os termos de bancos de dados, cada atributo (campo) da tabela passa a ser armazenado em uma tupla (registro) separada, de forma que os valores de cada atributo compõem uma tupla da tabela.

Dessa forma, os campos que compõem o registro lógico, ou seja, que possuem uma associação lógica entre si não são mais armazenados seqüencialmente em um mesmo registro físico. Como eles estão dispostos em registros diferentes, é necessário considerar algum mecanismo para manter um vínculo lógico eles. Para que esta associação lógica possa continuar existindo, consideram-se os dados como sendo de um mesmo registro lógico aqueles que estão armazenados na mesma posição lógica dentro de cada registro, conforme ilustra a Figura 14.

Para calcular corretamente a posição física de um determinado campo, deve-se considerar o tipo de dado que está sendo armazenado naquele registro físico e, de acordo com o espaço definido para o referido campo, pode-se calcular uma determinada posição. A este cálculo dar-se o nome de *offset*. Isto se torna necessário porque os campos de um registro lógico podem estar armazenados em posições físicas diferentes dentro de cada registro de dados, pois essa posição vai depender do tipo de dado e tamanho definido para cada campo. Por exemplo, na Figura 14, onde é destacado o segundo registro lógico do arquivo, o Campo 1 (código do cliente) está armazenado em posição física diferente da posição física em que está armazenado o Campo 2 (nome do cliente) no registro 2, que, por sua vez, está diferente da posição física do Campo 3 (endereço do cliente) no registro 3. Isto acontece porque os campos são de tipos diferentes e por isso podem possuir tamanhos diferentes. Porém, mesmo com posições físicas distintas para cada campo, a posição lógica deste registro lógico corresponde à segunda posição, pois este é o segundo registro lógico da estrutura de armazenamento.

A principal motivação para definir a idéia da proposta apresentada neste trabalho é a necessidade de criar um mecanismo mais abrangente que permitisse melhor gerenciar os dados em dispositivos móveis do tipo PDA associado à possibilidade de flexibilizar as estruturas de armazenamento no sentido de facilitar alterações dos registros de dados existentes nos arquivos através da evolução de esquemas de dados. Além disto, tal proposta oferece os fundamentos necessários para disponibilizar funcionalidades que garantam a consistência dos dados através de restrições de integridade e potencializar a utilização de técnicas de compressão de dados visando otimizar o espaço utilizado em memória.

Na forma inicialmente apresentada na Figura 12, não é possível alterar a estrutura de registro do arquivo após sua criação. Caso fosse necessário incluir um novo campo em um arquivo existente, seria necessário criar um novo arquivo com a estrutura de registro antiga adicionando o campo a ser incluído, em seguida, copiar os dados do antigo arquivo para o arquivo criado já com o novo campo definido e, por fim, excluir o arquivo com a estrutura antiga.

Na estrutura proposta, a necessidade de um novo campo em um determinado arquivo seria resolvida com a inclusão de um novo registro no arquivo, conforme Figura 15. Neste exemplo, o novo registro (Campo 4) irá armazenar os dados do campo referente ao bairro do cliente. O mesmo acontece para a necessidade de excluir um determinado campo de um arquivo, pois para fazê-lo basta realizar um comando de exclusão do registro que armazena os dados a serem removidos do arquivo.

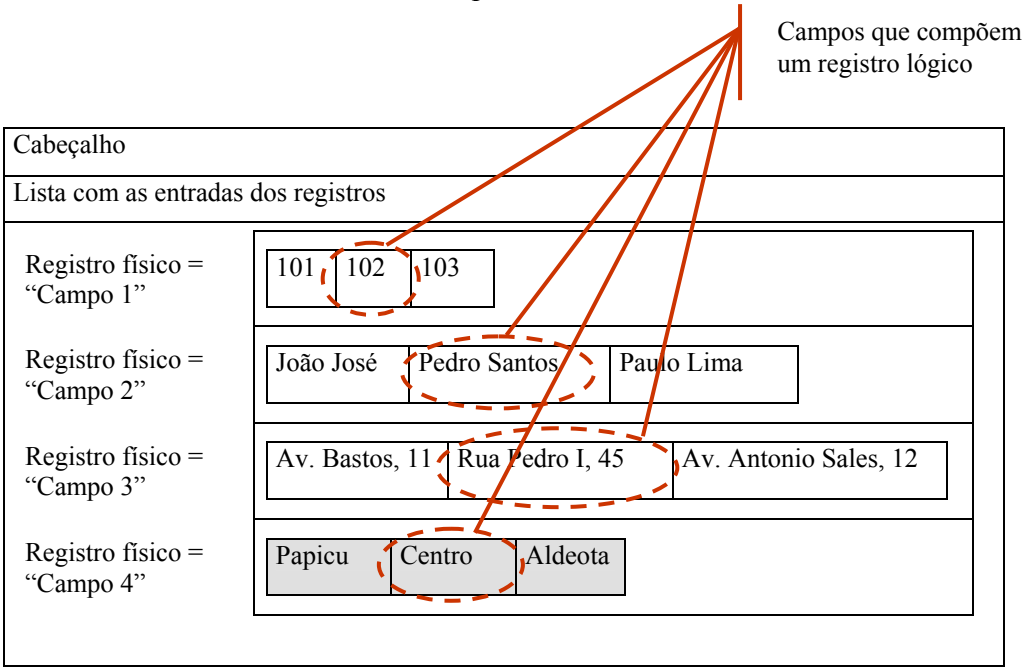

Figura 15 - Estrutura de armazenamento proposta com a inclusão de um novo campo de dados

Associando a estrutura de armazenamento proposta aos conceitos e características existentes nos sistemas de bancos de dados convencionais, é possível definir um mecanismo diferenciado de gerenciamento dos dados e dos esquemas que seja compatível com esta nova estrutura de armazenamento. Para tanto, pode-se considerar que uma tabela pode ser representada pelo arquivo que contém os registros de dados, de forma que os registros do arquivo armazenam os dados de cada coluna definida na tabela. Partindo deste princípio, pode-se trabalhar com os arquivos adicionando sobre estes uma camada de abstração que permite disponibilizar uma visão relacional dos dados armazenados.

Para disponibilizar estas funcionalidades é necessária a criação de estruturas de controles que armazenem informações dos metadados que possibilitam o gerenciamento das estruturas de armazenamento e dos dados armazenados nestas estruturas. Os arquivos que armazenam os dados de esquema guardam informações como: nome dos bancos de dados, nome das tabelas, nome das colunas, tipos de dados de cada coluna, identificação de chave-primária, identificação da possibilidade de valores nulos, tamanho dos campos definidos, relacionamentos entre tabelas, entre outras. Todas estas informações de controle são necessárias para prover as devidas condições para um gerenciamento adequado dos dados de forma a oferecer uma visão relacional utilizando os conceitos de tabelas e colunas de um sistema de banco de dados tradicional.

Uma outra motivação para se definir tal estrutura é a necessidade de otimizar o processamento já que se está considerando um ambiente com recursos computacionais limitados como memória e poder de processamento. A estrutura de armazenamento proposta apresenta ganhos de desempenho na execução de consultas quando são consideradas operações de projeção. Para acessar uma determinada coluna, será necessário apenas percorrer os dados armazenados no registro correspondente à coluna especificada, o que gera custo reduzido de processamento com a ausência de operações de acesso a todos os demais registros. Em contra partida, tal estrutura gera um custo adicional ao executar consultas com filtros horizontais (seleções) por ser necessário realizar cálculos para acessar diretamente os dados que atendem as condições da seleção. Mais adiante, é apresentada uma análise conceitual sobre os custos e benefícios desta estrutura proposta.

A Figura 16 ilustra a estrutura criada para armazenar os dados da tabela `tbCliente`. Nela cada registro armazena os dados correspondentes a um mesmo tipo de dado que representam as colunas da tabela. No exemplo, o terceiro registro da tabela indica que ele

armazena os dados da "`Coluna 3`" correspondente ao atributo `Endereço`. Assim, para acessar os dados de uma coluna específica (operação de projeção), basta percorrer o registro de dados que corresponde a coluna desejada. Como já mencionado, para esse tipo de consulta, a operação de busca torna-se mais rápida, uma vez que será necessário percorrer apenas um único registro de dados para recuperar as informações de um determinado atributo.

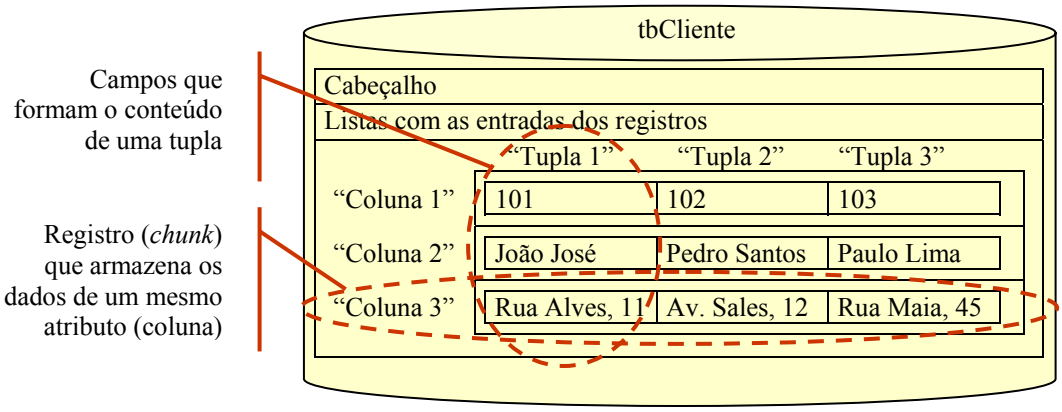

Figura 16 - Estrutura de armazenamento proposta considerando conceitos de bancos de dados

Dessa forma, para realizar uma determinada consulta na estrutura de armazenamento proposta é necessário acessar apenas os dados das colunas referenciadas na consulta. Este fato otimiza o processamento de busca uma vez que não é necessário acessar os dados de colunas que não são relevantes para a obtenção do resultado desejado.

Para demonstrar a eficiência da estrutura proposta, a seguir, é descrita uma análise conceitual sobre os pontos positivos e negativos da estrutura apresentada em relação à estrutura tradicional utilizada pelos sistemas de arquivos dos dispositivos móveis do tipo PDAs existentes. A proposta demonstra-se mais eficiente nos seguintes aspectos:

- A manutenção de esquemas das tabelas é extremamente facilitada. Para realizar a inserção ou exclusão de uma coluna (atributo) em uma tabela, será necessário apenas incluir ou excluir o registro correspondente à coluna. Essa característica permite que estruturas relacionais possam ser facilmente definidas e manipuladas inclusive em tempo de execução;
- Na criação de registros que armazenam dados referentes às colunas, o processo de execução de consultas com filtros verticais (projeções) é otimizado, pois a quantidade de registros acessados é diminuída. Isto se deve ao fato de que serão acessados apenas os registros com os dados referentes às

colunas que devem ser retornadas, ou seja, que fazem parte da projeção. Em um caso mais específico, quando uma operação de busca desejar retornar apenas os dados de uma coluna, será necessário percorrer apenas o registro correspondente à coluna especificada na operação de projeção. O ganho de performance em consultas com projeções é bastante relevante por se tratar de ambientes com recursos computacionais limitados em que uma pequena otimização no processamento de consultas pode gerar ganhos significativos no tempo de resposta das operações realizadas pelos usuários;

- Operações de agregação são otimizadas já que, geralmente, são aplicadas em um único atributo. Como os dados de um atributo estão armazenados seqüencialmente em um mesmo registro, para retornar o resultado de operações como `SUM`, `MAX` ou `MIN`, basta percorrer o registro especificado pela consulta aplicando a operação submetida;

- Facilidade na integração dos dados armazenados no dispositivo móvel quando estes forem enviados para um servidor de banco de dados do ambiente *desktop*, pois a consistência dos dados armazenados no dispositivo oferece menor probabilidade de problemas de migração e conflitos dos dados quando estes são sincronizados com o servidor de banco de dados central. Quando os dados são armazenados sem qualquer tratamento de consistência, como acontece ao trabalhar diretamente com o sistema de arquivos do ambiente, ao realizar a integração dos dados com um servidor central de banco de dados através do processo de sincronização, é maior a possibilidade da ocorrência de problemas relacionados à inconsistência dos dados. Problemas como restrições de chave e integridade referencial são apresentados apenas no processo de migração dos dados o que retarda a identificação destes tipos de problemas.

Muito embora tenham sido constatadas vantagens na estrutura de armazenamento proposta, esta abordagem pode apresentar desvantagem na execução de consultas sem filtros verticais (operação de projeção), onde se deseja recuperar todos os atributos de uma tabela. Neste caso, para recuperar uma determinada tupla com todas as suas colunas, será necessário o acesso a todos os registros na posição correspondente a tupla desejada. Por exemplo, na Figura 16, pode-se observar que para recuperar a "`Tupla 1`" é necessário acessar os registros referentes às colunas: (1) `Código`, (2) `Nome` e (3) `Endereço` que estão em registros distintos. Porém, como já mencionado, essa desvantagem é minimizada quando existem filtros

horizontais (operações de seleção), pois tais condições diminuem o domínio de busca da consulta submetida uma vez que a abordagem proposta utiliza o conceito de *offset*.

4.2.2 Mecanismo de Acesso a Dados

Quando uma consulta com filtros horizontais é submetida, inicialmente são tratados os filtros definidos. Assim, será feita uma busca seqüencial no(s) registro(s) que possui(em) o(s) atributo(s) utilizado(s) no(s) filtro(s). Um registro é acessado de forma integral e os dados armazenados no registro são tratados de maneira seqüencial. Sempre que um determinado dado atender a condição definida para aquele atributo, a posição deste dado no registro é armazenada em um vetor de índice. Este vetor de índice é definido como o vetor que irá armazenar todas as posições (índices) dos dados que atendem à condição do filtro, ou seja, que satisfazem à consulta. Para ilustrar, considere a Figura 17 como sendo um arquivo criado a partir da estrutura de armazenamento proposta.

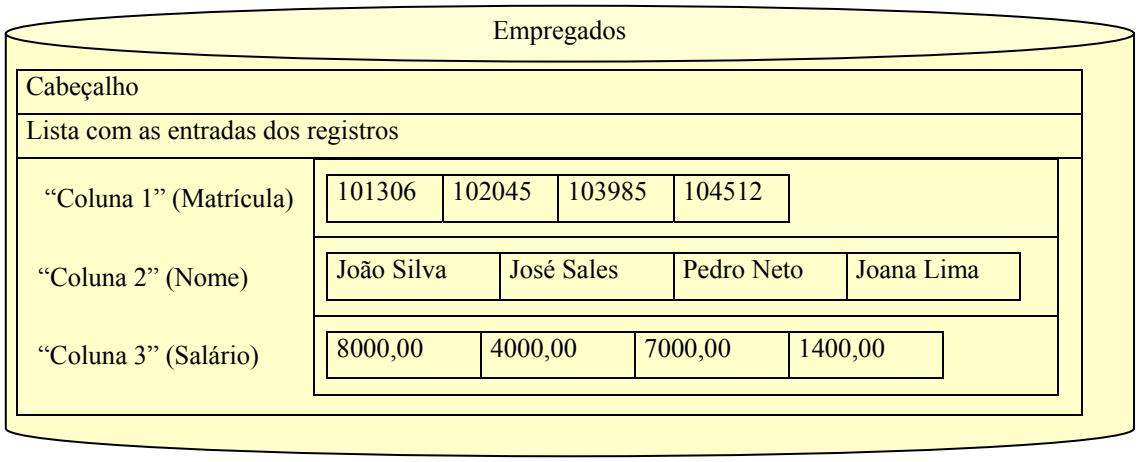

Figura 17 - Estrutura de armazenamento proposta

Para realizar uma consulta que retorne o `Nome` dos funcionários que possuem `Salario` maior que R$ 5.000,00, a seguinte consulta deverá ser submetida:

```
SELECT Nome FROM Empregados WHERE Salario > 5000
```

Inicialmente, será realizada uma busca seqüencial no registro que possui o filtro da consulta: o campo `Salario`. Para cada item de dados será realizado o teste da condição de busca. Caso o item de dado atenda a condição, sua posição no registro será armazenada no vetor de índice. Após percorrer todo o registro aplicando a condição de busca em cada item de dado, o vetor de índice será utilizado para que sejam recuperados apenas os dados existentes

nas posições identificadas pela condição de busca. De acordo com o exemplo da Figura 17, o vetor de índice deverá possuir apenas as posições 1 e 3. Diante destas posições, o mecanismo de busca irá recuperar os campos que devem ser retornados pela consulta considerando apenas as posições existentes no vetor de índice. Neste caso, a consulta irá retornar os nomes "João Silva" e "Pedro Neto" por corresponderem as posições 1 e 3 recuperadas anteriormente.

No caso de existirem vários filtros na consulta, será criado um vetor de índice para cada atributo pesquisado que armazenará as posições que satisfazem à consulta. Em seguida, as posições armazenadas no(s) vetor(es) de índice(s) serão tratadas de acordo com a operação lógica definida na condição da consulta. Ou seja, se existirem duas condições e um operador lógico `AND` entre elas, por exemplo, será feita uma consulta sobre as posições recuperadas e armazenadas nos vetores de índices de cada atributo para identificar apenas aquelas posições que estão presentes em ambos os vetores. O resultado desta operação lógica é armazenado em um novo vetor final chamado de vetor resultante. Este vetor irá armazenar os índices que realmente atendem a todas as condições definidas no filtro da consulta.

Concluída esta primeira fase de busca em que são identificadas todas as posições dos dados que satisfazem à consulta, o vetor resultante é utilizado para auxiliar a operação de *offset*. Esta operação permite que depois de identificadas às posições das tuplas (índices do vetor resultante) que atendem a condição da seleção, sejam acessados apenas os dados das colunas referentes a tais índices. Isto é possível porque para cada atributo a ser recuperado é identificado inicialmente o seu tipo e tamanho. Sabendo-se que todos os campos de um atributo são de tamanho igual, é possível acessar diretamente a posição de todos os índices identificados no vetor resultante através da operação de *offset*. Esta operação utiliza o valor referente ao espaço reservado em memória para determinado campo juntamente com o índice identificado anteriormente pelo vetor resultante para calcular a posição que deve ser acessada no registro e recuperar o valor do atributo desejado. A fórmula utilizada pelo *offset* é:

Offset = Tamanho do campo * (Índice recuperado do vetor resultante -1)

A operação de *offset* é executada para todos os índices do vetor resultante de forma que todos os dados que atendem a condição da consulta sejam recuperados através de um acesso direto em sua posição de memória. Dessa forma, a operação de *offset* é executada para todos os índices do vetor resultante para cada atributo que deverá ser retornado pela

consulta. Os dados recuperados são armazenados em uma estrutura temporária para depois serem apresentados como resultado da consulta. Dessa forma, a busca seqüencial acontece apenas nos atributos definidos na condição da operação de seleção.

No caso de operações de atualização e exclusão de dados, a operação de *offset* também é utilizada para identificar a posição de memória de uma determinada tupla. Adicionalmente, é realizada a operação de alteração no dado quando for submetida uma atualização, ou uma operação de exclusão do dado quando for submetida uma deleção. Quando acontece uma operação de exclusão, é realizado um deslocamento dos dados posteriores à posição de memória que teve seus dados excluídos, de forma que os dados imediatamente posteriores assumem a posição dos dados excluídos. Isto acontece para que os dados estejam sempre armazenados seqüencialmente no arquivo mantendo as condições necessárias para que a operação de *offset* possa sempre ser realizada adequadamente. Já a operação de inserção inclui os dados sempre no final de cada registro para todos os atributos da tabela.

Percebe-se dessa forma, que a arquitetura proposta facilita a evolução de esquemas em detrimento à exclusão de dados, uma vez que tais operações necessitam de um processamento adicional para redimensionar o tamanho do registro para evitar a desfragmentação de memória. Porém, o custo adicional existente nesta alteração de dados só é relevante quando são realizadas operações de exclusão que não possuem filtros horizontais, pois, dessa forma, todos os registros necessitam ser atualizados sem nenhuma restrição de condição.

4.2.2.1 Execução de Consultas com Filtros Horizontais (Operações de Seleção)

Para ilustrar um exemplo da execução de consultas com filtros horizontais, considere a tabela apresentada na Figura 18. Para realizar uma consulta para retornar todos os dados dos clientes cujo campo `Codigo` seja maior que `101`, a seguinte consulta deve ser definida:

```
SELECT Codigo, Nome, Endereco
FROM Clientes
WHERE Codigo > 101
```

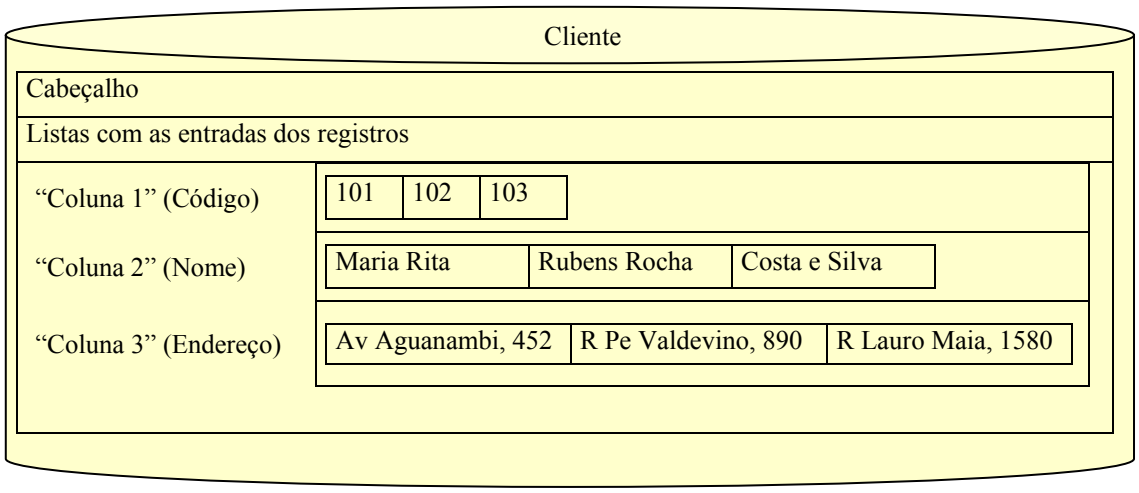

Figura 18 - Estrutura de armazenamento proposta com dados da tabela Cliente

O mecanismo de consulta irá percorrer somente o registro que faz parte da condição de busca, neste caso o registro que corresponde aos dados do campo `Código`, para identificar as posições das tuplas que satisfazem à condição de busca. O resultado dessa busca inicial é armazenado no vetor de índice. Assim, o vetor de índice irá armazenar as posições do atributo que satisfaz a condição de busca. Neste caso, o vetor de índice seria um vetor com duas posições como segue:

Vetor de índice:

2	3

No segundo momento, as posições armazenadas no vetor de índice são utilizadas para acessar as demais colunas que devem ser retornadas pela consulta. Esta segunda busca será feita nos demais registros considerando apenas as posições já identificadas que satisfazem à condição de busca. Assim, será acessado o registro correspondente ao nome do cliente nas posições 2 e 3. Tais dados serão armazenados em uma estrutura temporária para posterior apresentação. O mesmo acontecerá para o registro correspondente ao endereço do cliente. De forma que, ao finalizar a busca em todos os registros, será apresentada a estrutura que armazenou os resultados parciais contendo agora o resultado final da consulta inicialmente submetida, conforme Figura 19.

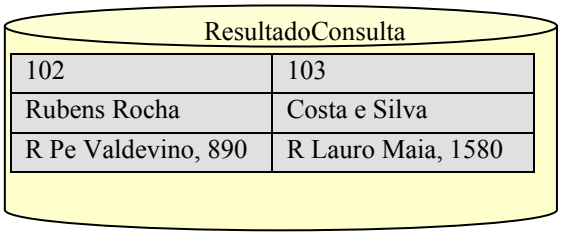

Figura 19 - Resultado da consulta submetida à tabela Cliente

A Figura 20 apresenta um outro cenário para a execução de uma consulta com filtros horizontais, em que, inicialmente, são tratadas as condições que devem ser atendidas pela consulta. O campo `Codigo` da tabela `tbCliente` é acessado e são identificados os índices que satisfazem à condição definida na operação de seleção, de forma que estes índices são armazenados no `Vetor de índice`$_1$. A segunda condição referente ao campo `Nome` também é tratada e os índices que satisfazem à restrição também são identificados e armazenados no `Vetor de índice`$_2$. Com base nos vetores de índices identificados é aplicado o operador lógico definido na consulta e um `Vetor resultante` é definido com os índices que satisfazem à todas as condições da consulta.

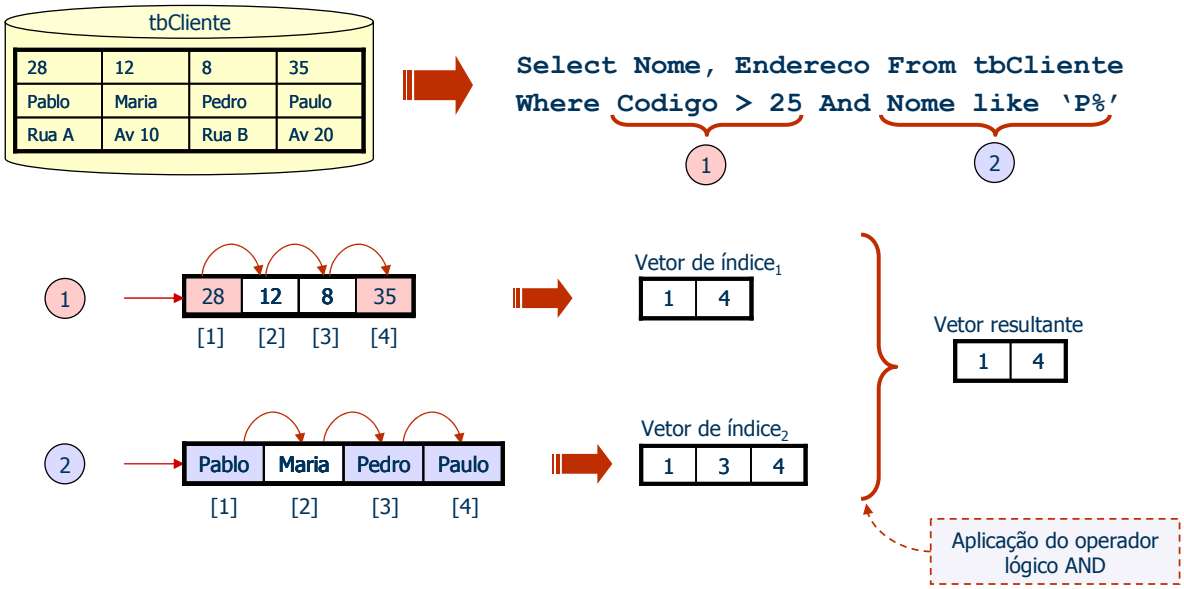

Figura 20 - Cenário inicial de uma consulta com filtros horizontais (seleções)

Conforme ilustra a Figura 21, a partir do `Vetor resultante` encontrado, é realizado o cálculo do *offset* para cada atributo que deve ser retornado pela consulta. São acessados os registros de dados correspondentes aos campos definidos na seleção, `Nome` e `Endereco`, e depois é calculada a posição de memória de acordo com os índices do `Vetor resultante`. Os dados existentes nestas posições retornadas pelo *offset* são armazenados em uma estrutura temporária para serem retornados como resultado da consulta inicialmente submetida.

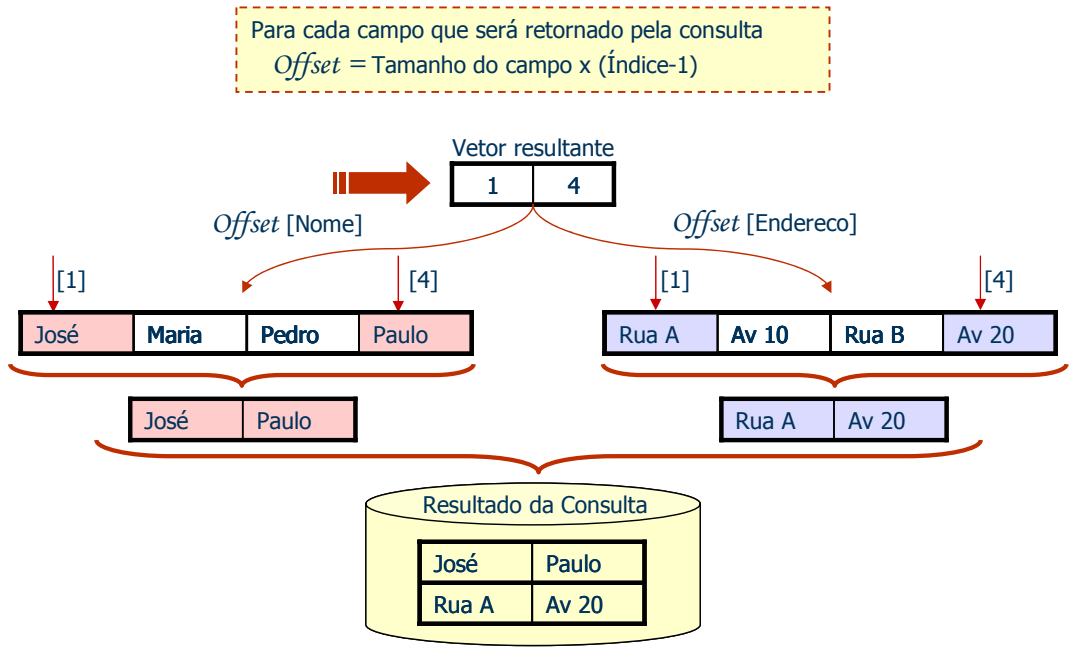

Figura 21 - Cenário final de uma consulta com filtros horizontais (seleções)

4.2.2.2 Consultas com Filtros Verticais (Operações de Projeção)

Para demonstrar a execução de consultas apenas com filtros verticais, a Figura 22 apresenta um cenário em que todos os dados de determinados atributos são retornados como resultado da consulta submetida.

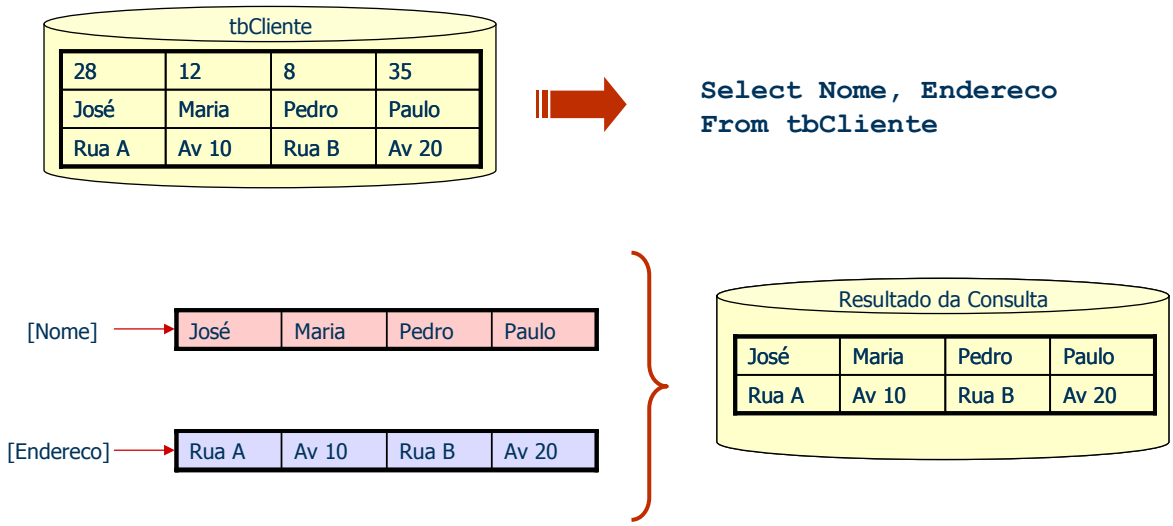

Figura 22 – Cenário de uma consulta com filtros verticais (projeções)

Para ilustrar um exemplo de consultas com operações de projeção associadas às condições de busca, a Figura 23 mostra um conjunto de dados armazenados em uma estrutura de arquivo representando a tabela de `Empregados`.

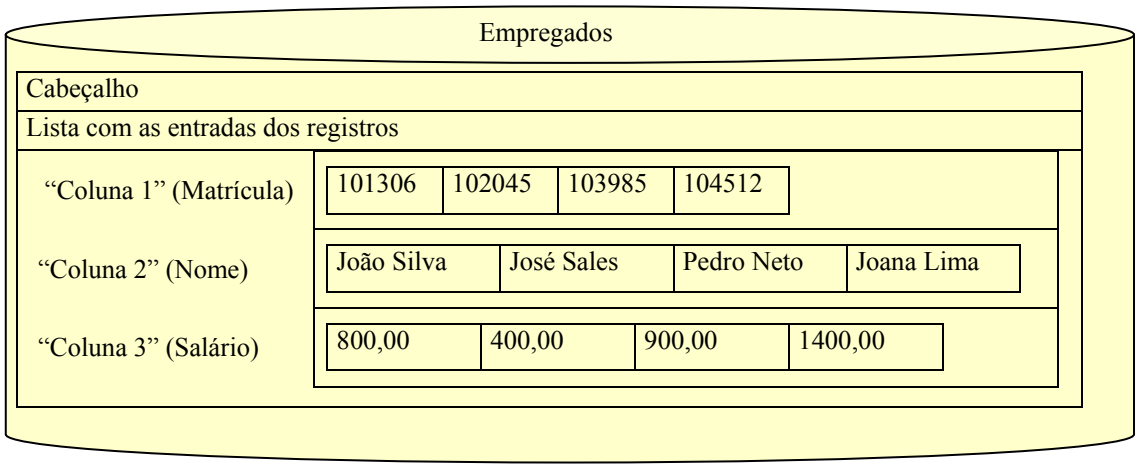

Figura 23 – Estrutura de armazenamento proposta com dados da tabela de Empregados

Para que sejam recuperados os dados referentes ao `Nome` e ao `Salario` dos empregados que recebem menos que `R$ 1.000,00` e têm seu começado pela letra `J`, a seguinte consulta deve ser definida:

```
SELECT Nome, Salario
FROM Empregados
WHERE Salario < 1000 AND Nome LIKE 'J%'
```

A execução desta consulta será realizada da seguinte forma:

a) Será feita uma busca no registro que contém os salários em que serão identificados os índices que possuem valor inferior a `1000`. Neste caso, o vetor seria o seguinte:

$Vetor\ de\ índice_1$:	1	2	3

b) Em seguida será tratada a segunda condição onde o nome deve começar com a letra `J`. O resultado desta segunda condição retorna os seguintes índices:

$Vetor\ de\ índice_2$:	1	2	4

c) Depois de identificados os índices que atendem a todas as condições da consulta, será realizado o tratamento destes dados considerando o operador lógico utilizado na consulta. Neste caso, será feita uma identificação de quais índices estão presentes em ambas as condições, visto que a consulta utilizou o operador lógico `AND`. Assim, o vetor resultante agora passará a armazenar apenas os índices que seguem:

Vetor resultante:

1	2

d) Apenas os índices 1 e 2 atendem a ambas as condições definidas na consulta, dessa forma, para retornar a projeção da consulta será necessário acessar apenas os registros (tuplas) 2 e 3 que contêm os dados `Nome` e `Salario` dos empregados, respectivamente, conforme mostra a Figura 24.

e) Para acessar o registro com os dados da coluna `Salario`, será utilizado o cálculo para identificar o *offset* a ser utilizado. Recupera-se o tamanho do campo da coluna na tabela que armazena o esquema e multiplica-se o valor do tamanho pelo índice identificado no vetor resultante menos 1. Este cálculo é feito para todos os índices retornados de forma que ao final obtêm-se todos os dados referentes ao nome do empregado que satisfazem à condição.

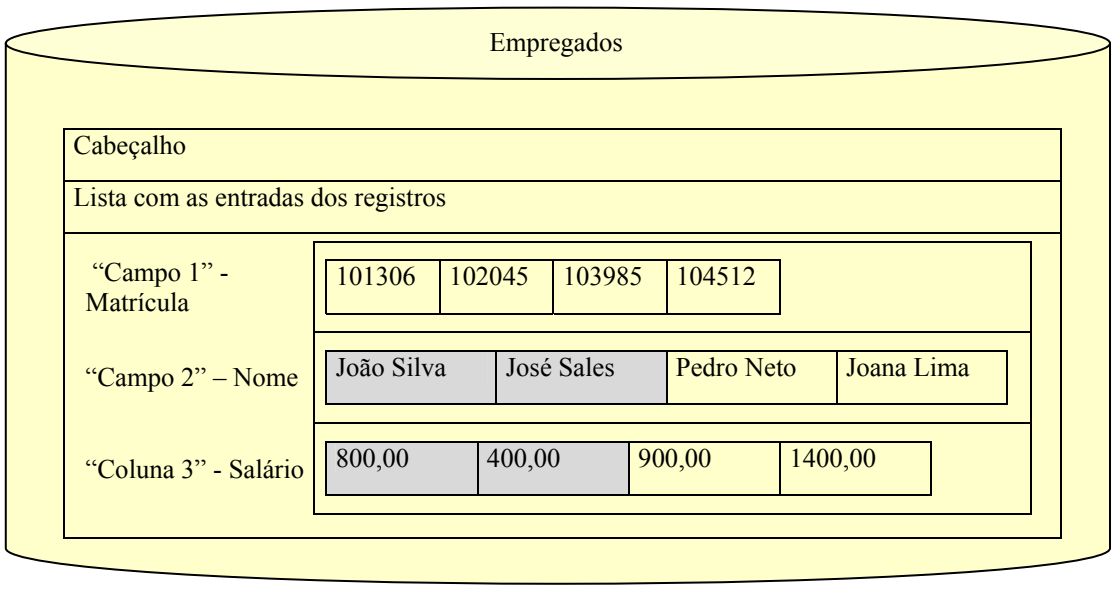

Figura 24 – Campos resultantes de uma consulta com filtros

f) Os dados resultantes desta primeira consulta são armazenados em uma estrutura temporária que, posteriormente, receberá os demais dados a serem retornados pela consulta.

g) Em seguida, serão recuperados os dados da coluna que armazena os nomes dos empregados. Será identificado o tamanho do campo `Nome` e será aplicado sobre este valor o número do índice que se deseja acessar. Os valores recuperados são adicionados na estrutura temporária como sendo outra coluna

de dados e quando associados formarão o resultado da consulta submetida, conforme Figura 25.

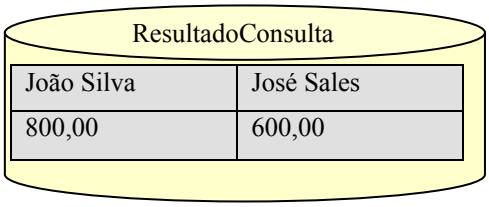

Figura 25 - Estrutura temporária criada para armazenar o resultado da consulta

h) Recuperados todos os atributos a serem retornados pela consulta, disponibilizam-se os dados com o resultado da consulta através da projeção de todos os dados armazenados na tabela temporária.

Uma vantagem da estrutura proposta é viabilizar a manipulação dos dados como se estes estivessem armazenados em bancos de dados relacionais, pois ela permite a criação de mecanismos que melhoram o gerenciamento dos esquemas, garantindo a consistência dos dados e propiciando uma maior eficiência em consultas com projeções verticais. Outro fator importante é que ao considerar que cada registro é formado por um mesmo tipo de dado, foi possível definir técnicas de compressão de dados com excelentes taxas de compressão sobre os dados armazenados nessa nova arquitetura. Este último tópico será discutido na seção 4.4.

4.2.3 Estruturas de Armazenamento do Esquema de Dados

Para oferecer uma visão relacional dos dados armazenados em arquivos, foi necessária a criação de estruturas de controle que permitissem armazenar informações das estruturas de armazenamento de dados, os chamados metadados. Com as informações dos metadados é possível realizar um gerenciamento adequado das estruturas de armazenamento de dados e, consequentemente, dos dados armazenados nestas estruturas.

As estruturas de armazenamento dos metadados possuem informações como:

- Nomes dos bancos de dados existentes;
- Quantidade de bancos de dados existentes;
- Nomes das tabelas;
- Números de colunas e de registros de uma determinada tabela;

- Nomes das colunas das tabelas, tipos dos atributos da tabela, atributos definidos como chave-primária, atributos definidos como chave estrangeira;
- Definição de estruturas de visões;
- Definição de estruturas que compõem integridade referencial entre tabelas;
- Estruturas com cadastro de usuário e senha de acesso.

Todas as informações dos metadados utilizam arquivos do tipo PDB com a finalidade específica para o armazenamento de dados de controle para disponibilizar o suporte necessário para a estrutura proposta. Estas estruturas de esquema são utilizadas para auxiliar na construção da camada de abstração dos dados armazenados em arquivos com o objetivo de oferecer funcionalidades essenciais de um sistema de banco de dados tradicional como:

- Banco de dados com um conjunto de tabelas, visões, usuários;
- Estruturas de tabelas com colunas;
- Definição de integridade de chave;
- Definição de integridade referencial;
- Consultas com junção entre tabelas;
- Estruturas de visões.

Ao criar um banco de dados, seu nome é gravado em um arquivo específico que armazena o nome de todos os bancos de dados criado seguindo a estrutura de armazenamento ilustrada na Figura 26. A partir desta estrutura inicial é possível controlar as demais estruturas associadas a cada banco de dados criado.

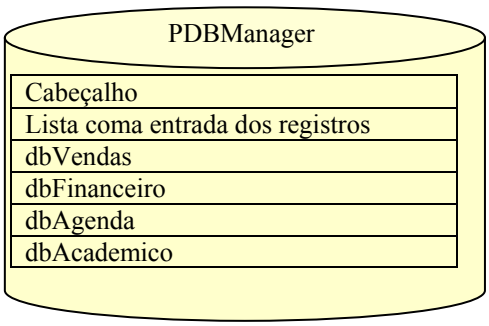

Figura 26 - Arquivo com os nomes dos bancos de dados existentes

Ao criar uma tabela, um novo arquivo é criado para armazenar os metadados das tabelas de um determinado o banco de dados. Para criar uma tabela é necessário definir o nome da tabela com as colunas que compõem a tabela (deve existir pelo menos uma coluna). Para definir uma coluna deve-se informar nome, tipo do dado, se permite nulo e se é chave-primária. Todos os metadados de uma tabela estão armazenados seqüencialmente no arquivo PDB cujo nome é o nome do banco de dados ao qual a tabela está associada. Cada registro deste arquivo armazena informações de uma tabela diferente, conforme mostra a Figura 27.

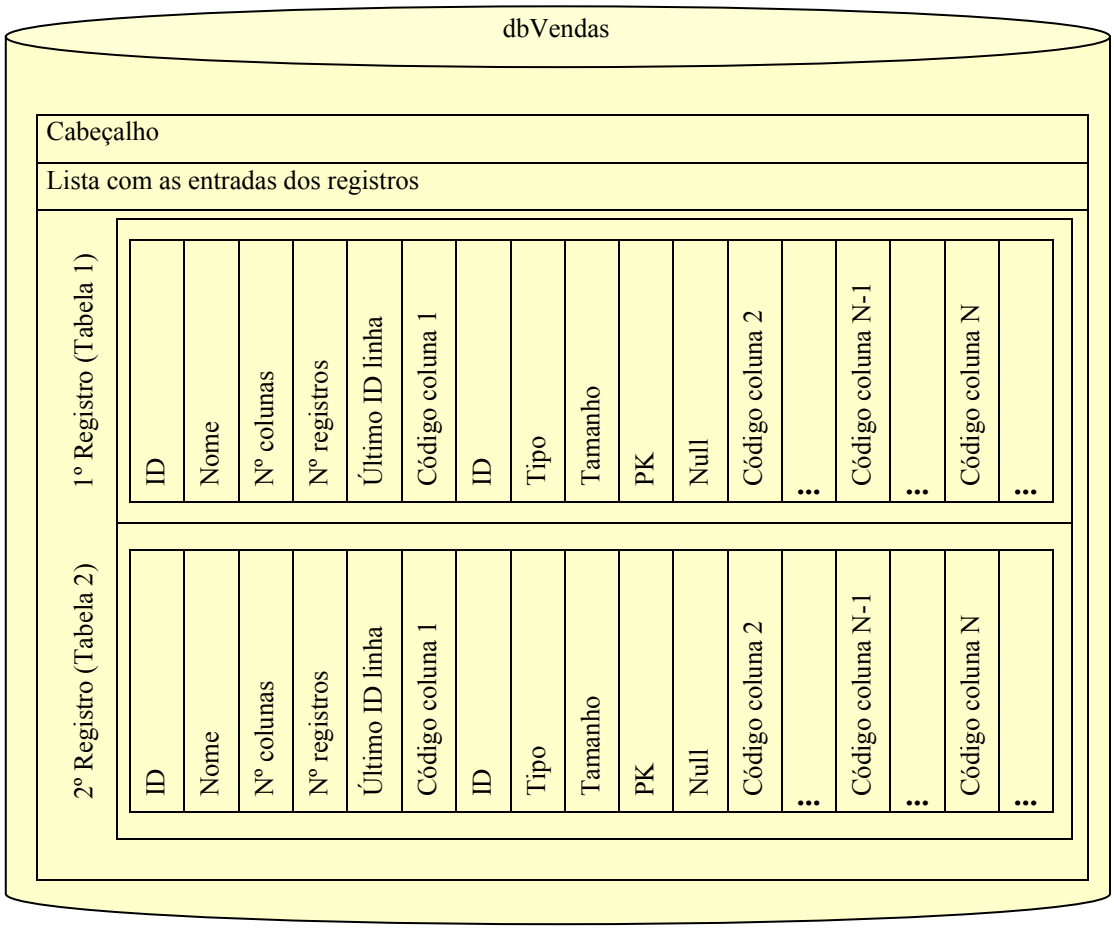

Figura 27 – Arquivo com os metadados das tabelas do banco de dados dbVendas

Dessa forma, os metadados que compõem o registro são armazenados seqüencialmente conforme segue: nome da tabela, identificador da tabela, número de colunas da tabela, número de registros da tabela, número da última da tabela. Em seguida e ainda no mesmo registro, seguem os dados das colunas que compõem a tabela: nome da coluna, o identificador da coluna, tipo de dado, tamanho que será reservado para o dado, se o campo é chave-primária e se permite valor nulo. Conforme a Figura 27, a partir deste ponto, o mesmo conjunto de metadados referentes à coluna será novamente armazenado para as demais

colunas da tabela de forma seqüencial e no mesmo registro. Para cada tabela do banco de dados será criado um registro seguindo esta estrutura definida.

As estruturas que armazenam os metadados das visões também são armazenadas em um PDB de forma seqüencial e seguem o seguinte esquema de armazenamento: nome da visão, nome da tabela, quantidade de colunas, nomes das colunas, quantidade de condições, as condições, indicação de junção e colunas que compõem a junção. Cada registro no arquivo de visões armazena os metadados de uma visão diferente, conforme apresentado na Figura 28.

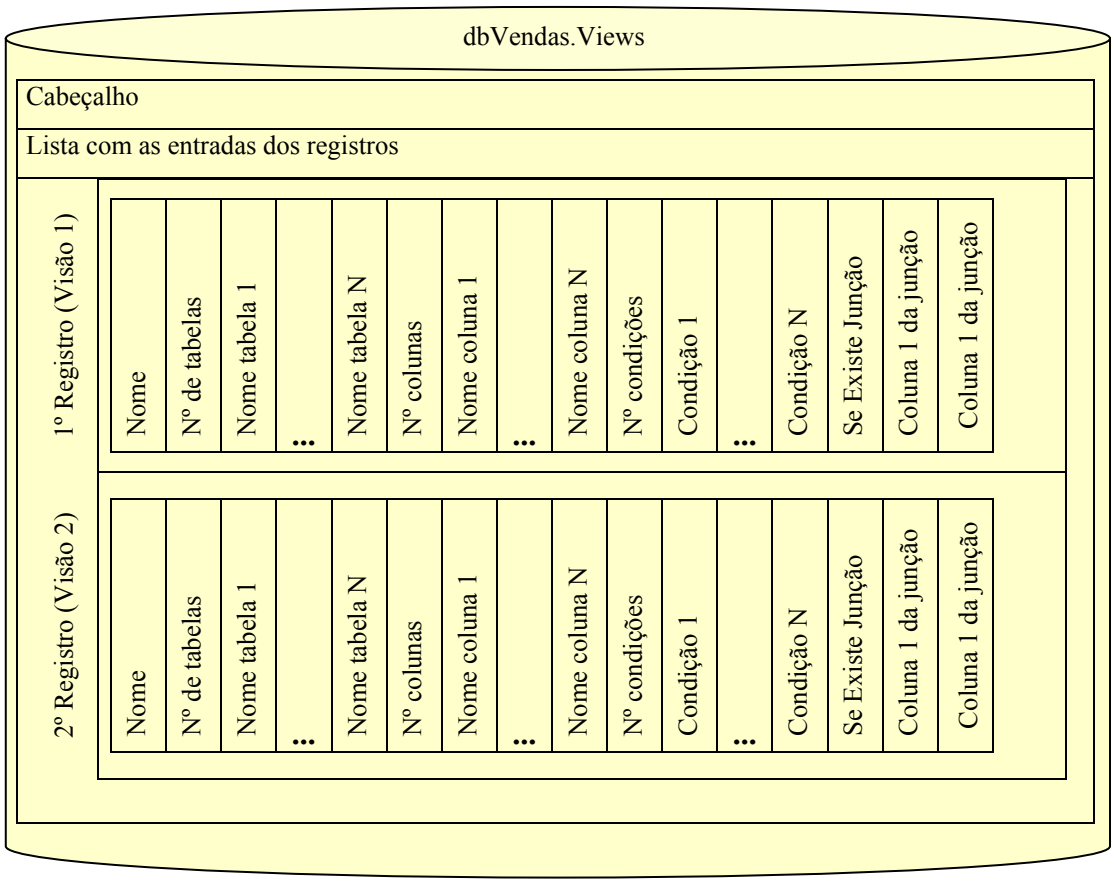

Figura 28 – Arquivo com os metadados das visões do banco de dados dbVendas

Também é criada uma estrutura de controle para guardar as definições de integridades referenciais entre as tabelas. Um arquivo é mantido para armazenar a definição de todas as integridades referenciais existentes em um determinado banco de dados. Cada registro do arquivo representa as definições de uma integridade referencial distinta de maneira que os metadados são dispostos seqüencialmente no registro da seguinte forma: nome do relacionamento, nome da tabela que referencia, nome da tabela referenciada, o nome do campo que referencia, nome do campo que é referenciado, conforme ilustra a Figura 29.

Todas as operações de atualização, inserção e exclusão de dados são realizadas considerando todas as restrições de chave e integridade referencial definidas pelo usuário.

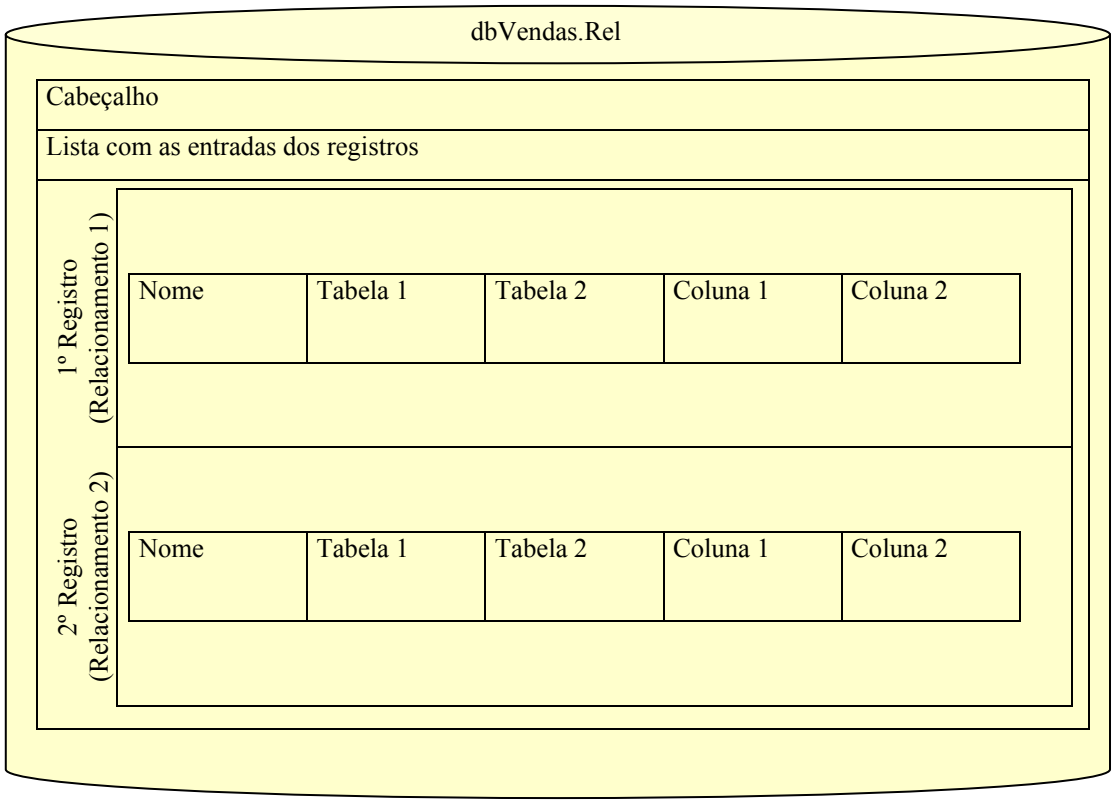

Figura 29 – Arquivo com os metadados dos relacionamentos entre as tabelas de um banco de dados

Existe uma outra estrutura definida apenas para armazenar metadados referente aos usuários. Tais metadados são armazenados nos registros do arquivo seqüencialmente da seguinte forma: nome do usuário e senha de acesso. A utilização de usuário e senha é um recurso que permite controlar o acesso aos dados de forma restrita às permissões cadastradas. Essa funcionalidade é útil quando existe um ambiente em que os dispositivos são utilizados por vários usuários de uma empresa, por exemplo. A senha é utilizada para viabilizar a proteção de um banco de dados de acordo com o usuário informado.

4.3 Gerenciamento dos Dados

Todo o gerenciamento criado a partir das estruturas de esquema definidas pela estratégia proposta fornecem o suporte necessário para disponibilizar as propriedades básicas de um sistema gerenciador de banco de dados para os arquivos existentes em um PDA.

As estruturas de controle criadas para um melhor gerenciamento dos dados e metadados são definidas e mantidas pelas funções existentes na API chamada *Database Library* (DBLib) que viabiliza a estruturação de toda a arquitetura proposta.

4.3.1 A Biblioteca *Database Library* (DBLib)

A biblioteca denominada *Database Library* (DBLib) foi desenvolvida para possibilitar o armazenamento e o gerenciamento de dados em PDAs de acordo com a estratégia proposta. A DBLib oferece o benefício de melhor gerenciamento dos esquemas de dados permitindo uma maior transparência no controle da consistência dos dados utilizados por uma determinada aplicação. A visão lógica oferecida pela DBLib tem por finalidade permitir a criação de aplicações mais flexíveis oferecendo uma camada de abstração dos dados armazenados fisicamente em memória e um conjunto de funcionalidades que permitem trabalhar utilizando os conceitos dos bancos de dados relacionais através das estruturas propostas na seção 4.2.1.

A biblioteca fornece funcionalidades para a criação de banco de dados, tabelas, colunas, definição de chave primária e integridade referencial. Também são disponibilizadas funções para manipular estas estruturas como os exemplos que seguem:

- A inclusão ou exclusão de uma coluna em uma tabela existente;
- A exclusão de uma tabela ou de um banco de dados;
- A definição de campo do tipo chave primária;
- A definição de integridade referencial entre as tabelas.

Além da possibilidade de criar e manipular os esquemas que definem as estruturas de armazenamento, esta biblioteca disponibiliza funções para manipular os dados armazenados em tais estruturas permitindo realizar operações de inserção, exclusão, alteração e busca dos dados armazenados nas tabelas.

4.3.2 Funcionamento das Estruturas de Controle

Na criação de um banco de dados a partir da DBLib, é criado um arquivo com o nome `PDBManager` considerado como o banco de dados principal, pois é a partir dele que todo o gerenciamento é iniciado, conforme ilustra a Figura 30. Neste arquivo serão armazenados os nomes de todos os bancos de dados criados utilizando a DBLib. Quando um

banco de dados é criado, o nome do novo banco de dados é inserido no arquivo `PDBManager`. Em seguida, é criado um outro arquivo com o nome do banco de dados criado (por exemplo, o arquivo `dbVendas` mostrado na Figura 31), que irá armazenar os metadados (dados do esquema) do banco de dados criado.

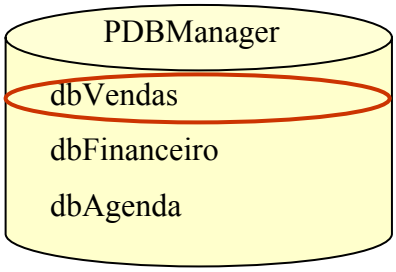

Figura 30 - Estrutura que armazena os nomes dos bancos de dados

No exemplo da Figura 31, O arquivo `dbVendas` armazena todas as informações relacionadas às estruturas das tabelas deste banco de dados. Cada registro deste arquivo corresponde às informações do esquema de uma determinada tabela, conforme já descrito na seção 4.2.3. O arquivo `dbVendas` possui quatro registros: o primeiro corresponde aos metadados da tabela `tbCliente`; o segundo, aos metadados da tabela `tbProduto`; o terceiro, aos metadados da tabela `TbPedidos` e o quarto, aos metadados da tabela `TbDetPed` (detalhamento do pedido). Portanto, cada registro do arquivo `dbVendas` armazena os dados que definem cada tabela criada nesse banco de dados como: o nome, o tipo e o tamanho de cada coluna, bem como se a coluna é uma chave primária ou se ela permite armazenar valores nulos. Além dos dados referentes às colunas, cada registro também armazena informações como número de colunas e número de tuplas de cada tabela.

Para garantir restrições de integridade como restrições de chave e de integridade referencial são utilizados os dados dos esquemas das tabelas que definem quais colunas possuem tais restrições. Da mesma forma, o armazenamento dos dados depende das restrições de domínio existentes nos metadados. Todas estas informações são sempre consultadas ao manipular os dados de uma tabela.

Conforme ilustra a Figura 31, o arquivo com os metadados de um banco de dados é o arquivo mestre para as demais ações a serem realizadas nas tabelas do banco de dados. Através deste arquivo é possível garantir a consistência dos dados armazenados nas demais tabelas do banco de dados.

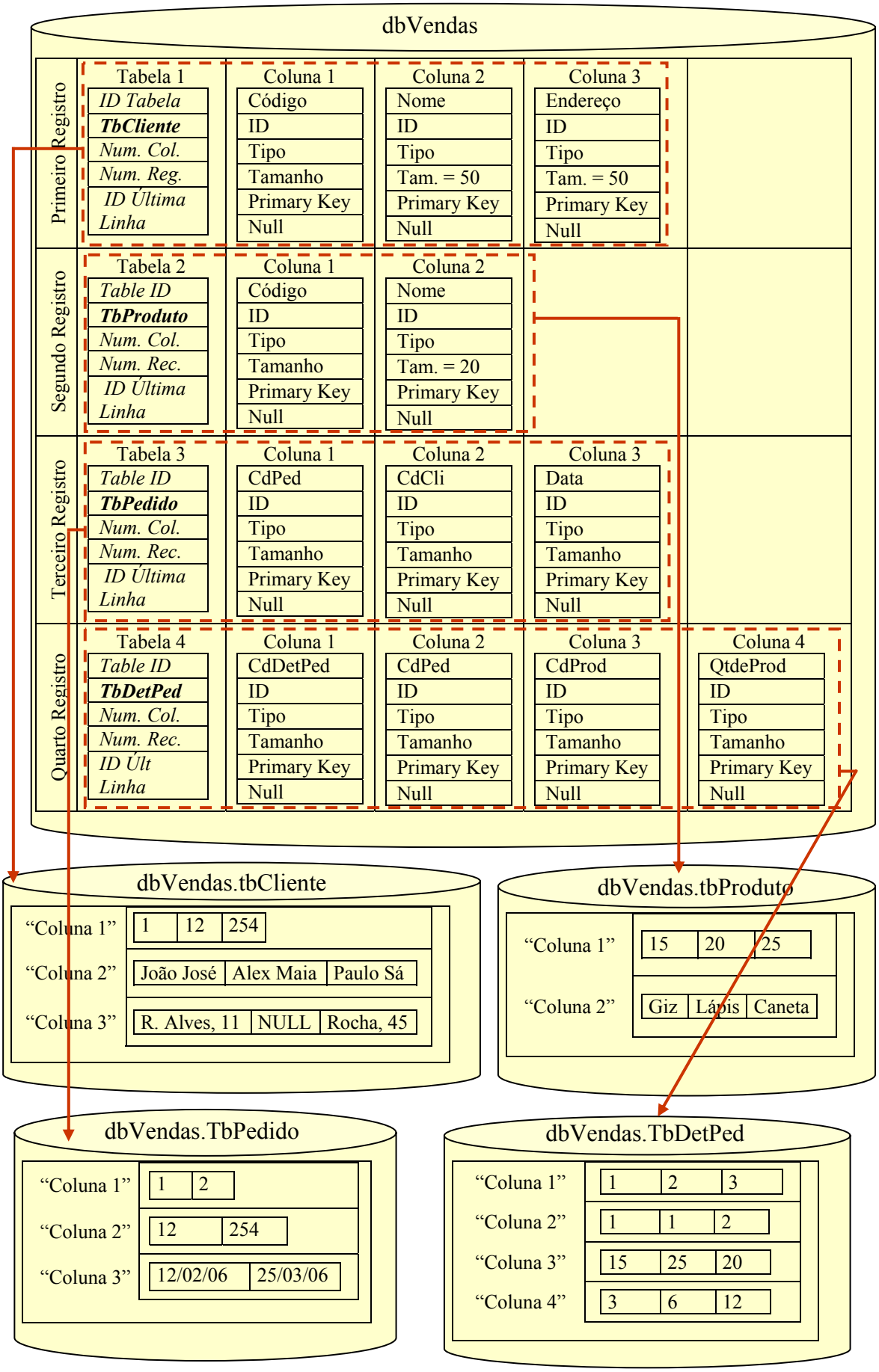

Figura 31 - Estrutura de gerenciamento dos arquivos criados a partir da DBLib

Ao criar uma tabela em um banco de dados, um arquivo será criado com o nome do banco de dados seguido do nome da tabela. Cada registro deste novo arquivo armazenará os valores referentes a uma coluna (atributo) da tabela, de forma que os dados da primeira coluna ficarão armazenados seqüencialmente no primeiro registro e assim sucessivamente. No exemplo da Figura 31, o arquivo `dbVendas.tbClientes` é criado para armazenar os dados da tabela `tbCliente` do banco de dados `dbVendas`. Cada registro do arquivo `dbVendas.tbClientes` armazena os dados referentes a uma coluna desta tabela.

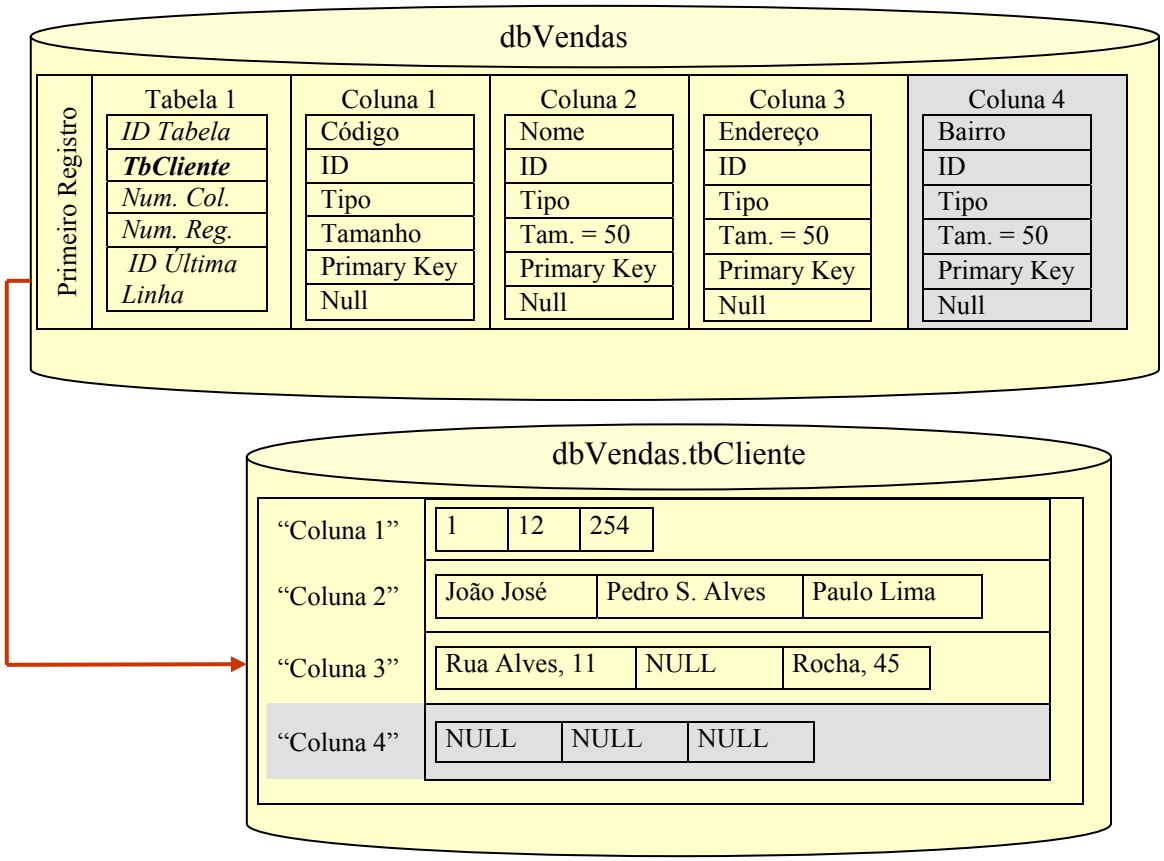

Figura 32 - Alteração no esquema de dados da tabela TbCliente

Dessa forma, a manutenção do esquema de dados é facilitada, pois para inserir uma nova coluna em uma tabela é necessário acrescentar a definição da nova coluna no registro referente à tabela a ser alterada. Por exemplo, para incluir a coluna `Bairro` na tabela `TbCliente` do banco de dados `dbVendas`, conforme mostra a Figura 32, basta executar o comando de alteração de tabela informando a estrutura do novo atributo. O registro do banco de dados `dbVendas` referente à tabela `TbCliente` será alterado recebendo os metadados do atributo a ser incluído na tabela. Após alterar o esquema, a DBLib se encarrega de também alterar a estrutura do arquivo `dbVendas.tbClientes`, que armazena os dados da tabela

`TbCliente`, incluindo um novo registro de dados referente ao novo atributo criado na tabela. A ação de exclusão de coluna de uma tabela acontece de forma semelhante, de forma que serão alterados os metadados da tabela e a estrutura de armazenamento do arquivo de dados da referida tabela. Neste caso, serão excluídos os metadados da tabela de esquema e o registro referente àquela coluna também será excluído da tabela que armazena os dados. As alterações dos esquemas de dados são sempre refletidas nas estruturas de armazenamento os dados.

Além das estruturas que armazenam os dados das tabelas, também poderá ser criado um arquivo com os relacionamentos, caso existam restrições de integridade entre tabelas. Conforme mostra a Figura 33, este arquivo armazenará os dados referentes às integridades referenciais entre duas tabelas existentes em um determinado banco. Cada registro desse arquivo corresponderá a um relacionamento contendo informações como os nomes das tabelas relacionadas, o nome do relacionamento e dos campos que fazem parte da restrição de integridade.

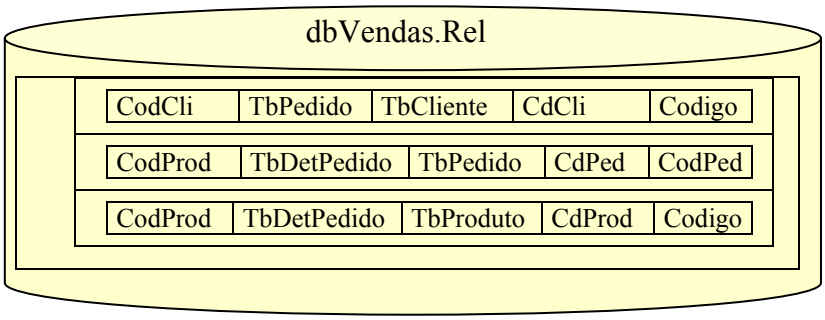

Figura 33 - Estrutura que armazena as integridades referenciais entre as tabelas

No exemplo da Figura 33, foram definidas três integridades referenciais. A primeira, definida entre a tabela `TbCliente` e `TbPedido`, indica a referência do campo `CdCli` da tabela `TbPedido` com o campo `Codigo` da tabela `TbCliente`. O segundo registro indica uma referência entre a tabela `TbDetPedido` e `TbPedido`, com os campos `CodPed` da tabela `TbDetPedido` e o campo `CodPed` da tabela `TbPedido`. Por fim, o terceiro registro define uma integridade referencial entre o campo `CodProd` da tabela `TbDetPedido` e o campo `Codigo` da tabela `TbProduto`. Dessa forma, é possível manter a integridade referencial entre as tabelas utilizando os relacionamentos apresentados na tabela `Rel` de cada banco de dados.

4.4 Compressão dos Dados

A utilização de técnicas de compressão em um sistema de bancos de dados oferece duas importantes vantagens: (1) redução dos custos de armazenamento e (2) economia de largura de banda de entrada e saída (E/S). Enquanto a primeira vantagem é um benefício mais óbvio da estratégia de compressão, a segunda vantagem decorre da diminuição da quantidade de dados que precisam ser movidos pelas operações no banco de dados. Por outro lado, a compressão pode causar um considerável *overhead* de CPU devido à necessidade de comprimir e descomprimir os dados a cada vez que estes forem utilizados, podendo aumentar consideravelmente o tempo de respostas de certas operações. Portanto, uma idéia importante é manter os dados em formato comprimido o maior tempo possível e só descomprimí-los quando e se for realmente necessário. Além disso, a utilização de técnicas de baixo custo computacional pode melhorar significantemente o tempo de resposta.

A natureza do processamento de consultas em banco de dados impõe diversas restrições na escolha do método de compressão mais adequado, o qual deve possuir duas características importantes: (1) ser de granularidade fina visando permitir o acesso aleatório a pequenos blocos de dados, e (2) ser extremamente rápido na compressão e acesso aos dados [6].

Métodos de compressão clássicos como a codificação Huffman [11], a codificação aritmética [46] ou a Lempel-Ziv [48] não são suficientemente rápidos, pois apresentam um *overhead* de processamento que anula os ganhos de performance obtidos. Tais métodos são eficientes apenas sobre grandes blocos de dados, sendo incompatíveis, portanto com acessos aleatórios a pequenas quantidades de dados.

A codificação baseada em dicionário consiste na substituição de cada *string* por um valor que a identifica unicamente. Tal associação é armazenada em uma tabela separada chamada dicionário. Os métodos de compressão baseados em dicionário podem ser estáticos (a tabela é construída antecipadamente), semi-estáticos (a tabela é construída durante o pré-processamento e fixada em seguida) ou adaptativos (a tabela é construída durante a compressão). Os métodos adaptativos são eficientes apenas quando aplicados sobre grandes blocos de dados [6]. Mesmo quando utilizados com a granularidade de nível de página, tais métodos ainda não são adequados porque para efetuar o acesso a uma única tupla precisam descomprimir uma página inteira.

Devido à heterogeneidade de tipos dos atributos em um banco de dados, o método de compressão mais adequado pode ser diferente para cada tipo de atributo. No entanto, no caso dos bancos de dados relacionais convencionais, é necessário que a mesma técnica de compressão seja aplicada em uma coluna (atributo) de uma tabela. Essa mesma técnica pode ser a mais adequada para um determinado atributo e a menos adequada para um outro atributo pertencente à mesma tupla. Considerando-se este detalhe, percebe-se que se os atributos forem todos do mesmo tipo, é possível aplicar a melhor técnica de compressão visando atingir igualmente todos os dados da tupla.

Assim, é possível perceber uma outra vantagem da arquitetura proposta, já que a mesma permite que sejam aplicadas técnicas de compressão diferentes para cada tipo de atributo. Isto é possível porque cada atributo é armazenado em uma tupla distinta de forma que uma tupla é formada pelo mesmo tipo de dado, pois armazena os dados de uma mesma coluna da tabela. De acordo com o tipo de dados definido para cada atributo, uma técnica de compressão mais eficiente para o tipo de dado em questão poderá ser aplicada. Essa característica permite maximizar as vantagens da técnica mais adequada para cada tipo de dado distinto de forma a obter melhores taxas de compressão de dados.

Dessa forma, o ganho com a otimização de técnicas de compressão (vantagem decorrente da forma de armazenamentos dos dados baseado em colunas) seria um fator compensatório para as desvantagens encontradas na execução de consultas sem filtros verticais apresentadas pela arquitetura proposta, visto que aspectos relacionados à redução no custo de armazenamento são fatores importantes para ambientes com recursos computacionais limitados do tipo PDA.

As estratégias de compressão apresentadas nesta seção são uma extensão baseadas na estrutura de armazenamento proposta neste trabalho. Tais técnicas são definidas em [4] e constituem o trabalho de dissertação do Ricardo Wagner C. Brito. Apesar destas técnicas de compressão não fazerem parte das contribuições do presente trabalho, elas são descritas a seguir para melhor contextualizar o leitor sobre aspectos relacionados à arquitetura que se referem ao armazenamento de dados em formato de compressão. Da mesma forma, a explanação de tais técnicas facilita o entendimento dos resultados obtidos nos experimentos apresentados neste trabalho na seção 5.3 que consideram os dados armazenados de forma comprimida. Nas subseções a seguir, são abordadas as técnicas de compressão desenvolvidas baseadas na arquitetura de armazenamento proposta neste trabalho.

4.4.1 Estratégias de Compressão dos Dados em PDAs

A arquitetura de armazenamento proposta neste trabalho ofereceu o suporte necessário para a aplicação eficiente de técnicas de compressão de dados para reduzir custo de armazenamento, permitindo realizar consultas sobre dados no formato de compressão e atingir altas taxas de compressão.

Na estratégia de compressão proposta foram aplicadas as técnicas de compressão numérica, compressão de booleano e compressão de *string* que serão discutidas a seguir.

4.4.1.1 Compressão Numérica

A idéia principal desta técnica é utilizar a menor quantidade possível de *bytes* para armazenar os valores inteiros. Por exemplo, o valor inteiro 1 só precisaria de 1 *bit* para ser gravado enquanto que o valor 100 precisaria de 7 *bits*. No entanto, o esquema de codificação proposto trabalha em nível de *bytes*. Esse "alinhamento a *byte*", apesar de representar uma menor economia de espaço, resulta em uma maior velocidade de acesso aos dados e foi escolhido por apresentar uma melhor relação custo-benefício.

Assim, tanto o inteiro 1 quanto o inteiro 100 poderiam ser armazenados em um único *byte*, o que representa uma economia de espaço em comparação com a maioria dos sistemas, onde os inteiros são armazenados usando 4 *bytes*. Um ponto chave da compressão numérica é a necessidade de armazenamento do número de *bytes* utilizados para armazenar os valores, pois esta informação será necessária no momento do acesso aos dados.

4.4.1.2 Compressão de Booleano

A idéia da compressão de tipo Booleano é semelhante a da compressão numérica, armazenar os valores no menor espaço possível. Na compressão numérica utiliza-se o *byte* como unidade mínima de armazenamento. Para valores Booleanos podemos diminuir ainda mais esta unidade. Como um dado deste tipo só pode assumir os valores '0' ou '1' (o valor `NULL` não é permitido para este tipo nesta proposta) oito valores de tipo Booleano podem ser guardados em um único *byte*. Isto representa uma grande economia de espaço em relação a uma abordagem convencional onde um valor Booleano ocuparia 1 *byte*.

4.4.1.3 Compressão de String

Os campos do tipo *string* são, na prática, representados com o tipo `CHAR`. Tais campos têm seu tamanho fixado na criação da tabela. Quando valores do tipo `CHAR` são armazenados, eles são preenchidos à direita com espaços vazios até o tamanho especificado. Dessa forma, o espaço alocado independe do valor a ser inserido. Isto representa um claro desperdício de armazenamento. Portanto, uma estratégia mais eficiente é ajustar o tamanho do espaço para armazenar o atributo de acordo com o valor a ser armazenado. Esta estratégia permite a especificação de *strings* com o tipo `VARCHAR`.

Assim, os valores são armazenados usando apenas quantos caracteres forem necessários. Se o dado a ser inserido é, por exemplo, a palavra "banco de dados" em uma coluna com tamanho declarado de 255 caracteres, apenas 14 *bytes* serão alocados (não haverá a inclusão de espaços vazios à direita da *string*). Além disso, assim como acontece para valores inteiros, o tamanho da *string* precisa ser gravado porque será necessário no momento da leitura do dado.

4.4.2 Acesso a Dados Comprimidos em PDAs

A estratégia de compressão utilizada tem como granularidade colunas (atributos) de uma tabela. Dessa forma, uma coluna é comprimida e descomprimida individualmente, sem a necessidade de ler ou atualizar outras colunas da tabela. Para uma consulta que, por exemplo, deseja retornar os campos `Matricula` e `Nome` de todos os empregados cujo `Salario` seja superior a `R$ 10.000,00`, o código em SQL seria:

```
SELECT Matricula, Nome FROM Empregado WHERE Salario > 10000
```

Neste caso, somente as colunas `Matricula`, `Nome` e `Salario` seriam acessadas e descomprimidas. Se para a tabela `Empregado` existirem ainda outras colunas como `Departamento`, `Endereco` ou `Telefone`, por exemplo, nenhuma delas seria acessada. Dessa forma, somente as colunas que serão utilizadas na consulta serão descomprimidas. A forma de acesso aos dados comprimidos vai depender do tipo do valor a ser lido. A seguir, serão descritas as técnicas para se recuperar dados comprimidos.

4.4.2.1 Dados do Tipo Inteiro

Devido ao "alinhamento a *byte*", um inteiro comprimido só pode ter 1, 2 ou 4 *bytes* (referentes aos tipos *char*, *short* ou *long*). Dessa forma, pode-se armazenar essa informação utilizando apenas dois *bits* tanto para campos que aceitam valor `NULL` quanto para os campos com valor `NOT NULL`, conforme mostra a Tabela 7. Em campos `NOT NULL`, o código "00" não é utilizado.

Tamanho (em *bytes*)	Código (par de *bits*)
0	00
1	01
2	10
4	11

Tabela 7 - Codificação do tamanho de inteiros

Visando uma maior compressão, os pares de "*bits* de controle" serão armazenados em grupos de quatro nos "*bytes* de controle" sendo inseridos a partir da esquerda (*bit* de mais alta ordem) no *byte*. Os *bytes* de controle formarão os registros de controle, conforme ilustrado na Figura 34. Com esta abordagem, cada *byte* de controle guarda a informação referente a quatro campos de uma coluna.

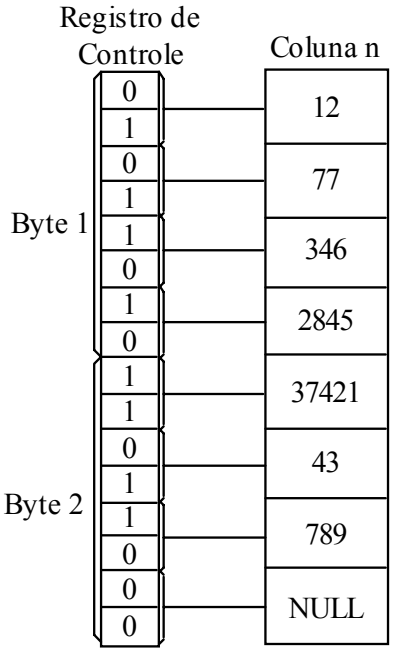

Figura 34 - Estrutura de um registro de controle para uma coluna do tipo Inteiro

Os registros de controle também serão utilizados para se calcular o *offset* de cada valor armazenado no registro de dados. Os valores dos *bytes* serão somados para se calcular a posição de um determinado dado.

4.4.2.2 Dados do Tipo Booleano

Conforme discutido anteriormente, nesta proposta um único *byte* pode comportar oito valores do tipo Booleano. Além disso, como o tamanho desses valores é fixo (sempre 1 *bit*) não há necessidade da utilização de um registro de controle para guardar essa informação.

Os valores booleanos são inseridos no *byte* a partir do *bit* mais à esquerda (de maior ordem). No momento da escrita ou leitura de um determinado valor, é necessário passar apenas a informação de sua coluna e de sua tupla. Com base nestes dados, podemos gravar ou recuperar o *bit* referente ao valor.

4.4.2.3 Dados do Tipo String

Os campos do tipo `CHAR` e `VARCHAR` são armazenados de uma maneira parecida com os tipos Inteiros. A parte referente ao seu valor é gravada com um tamanho variável e vai ocupar apenas o espaço mínimo necessário. Além disso, é preciso guardar a informação de seu tamanho que será utilizada no momento de acessar o dado. Esta informação será armazenada em um outro registro. Assim como nos valores inteiros, será utilizado o mínimo de espaço possível para guardar esta informação.

No caso de dados do tipo *String*, seu tamanho poderá variar, nesta arquitetura, de 0 a 255 caracteres. Essa informação poderá ser armazenada em um único *byte* de controle. Com isso, o registro de controle para valores do tipo *String* terá tantos elementos quanto o registro de dados, na proporção de um *byte* de controle para cada *string* armazenada na coluna. Um valor `NULL` poderá ser representado com o tamanho zero. A Figura 35 mostra um exemplo dessa estrutura.

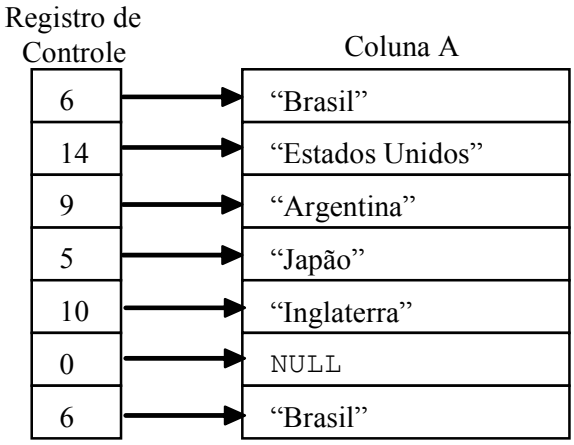

Figura 35 - Estrutura de uma coluna do tipo *String* e seu registro de controle

4.4.2.4 Estrutura dos Arquivos

Nesta seção será descrita a estrutura dos arquivos necessária para garantir a implementação da abordagem proposta de armazenamento e acesso de dados comprimidos.

Para cada coluna haverá um registro de controle associado. Para facilitar o acesso a estas informações, os registros de controle serão armazenados no mesmo arquivo dos registros de dados, alternando entre os dois tipos de registro, criando assim uma relação de paridade entre eles, conforme ilustrado na Figura 36. No entanto, somente para as colunas dos tipos Inteiro e `CHAR` as informações desses registros serão utilizadas. Para as colunas dos outros tipos, o registro de controle será criado apenas para manter a paridade e facilitar o acesso aos dados. Essa organização torna bastante simples o acesso às informações de controle ou aos dados propriamente ditos com base apenas no índice da coluna.

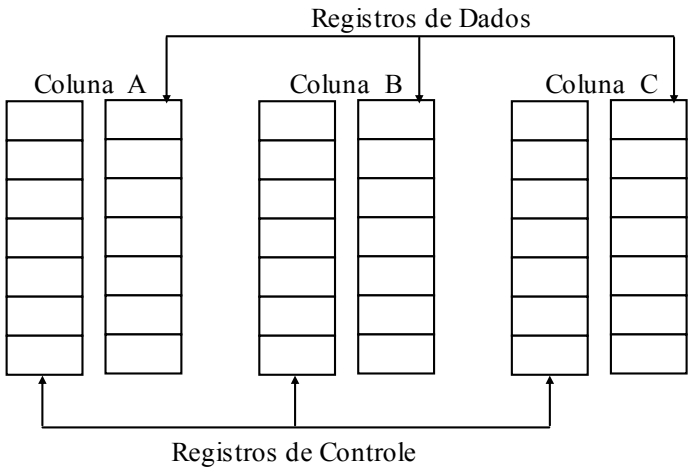

Figura 36 - Estrutura de um arquivo com os registros de controle e de dados de uma tabela

4.4.3 Gerenciamento dos Dados Comprimidos

As estratégias de armazenamento de dados definidas pela DBLib associada às técnicas de compressão de dados definidas pela CompLib são responsáveis por projetar uma estrutura de armazenamento capaz de manipular os dados em formato de compressão. Dessa forma, estruturas de controle necessitam ser criadas nos arquivos de dados para permitir que o armazenamento, o acesso e o gerenciamento dos dados possam ser realizados de forma comprimida. Essa nova estrutura é definida conforme ilustra a Figura 37.

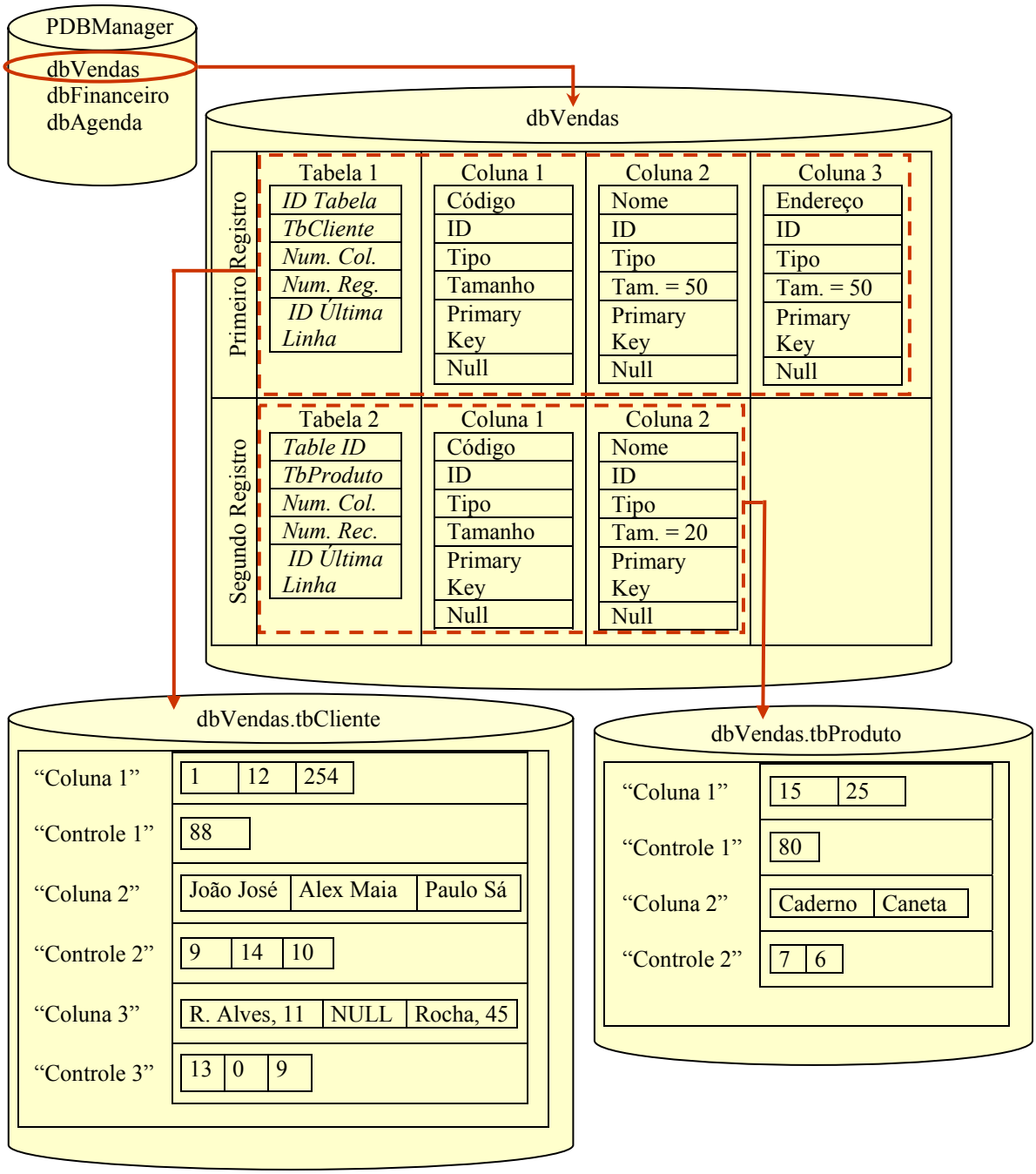

Figura 37 - Estrutura de armazenamento dos dados criada a partir da DBLib e da CompLib

4.5 Trabalhos Relacionados

A maioria dos SGBDs existentes atualmente no mercado (por exemplo Oracle, IBM DB2, Microsoft SQL Server) implementam o sistema de armazenamento de dados orientado a tupla, onde os valores de diferentes atributos de uma mesma tupla são armazenados sucessivamente em disco. Como apresentado, este trabalho apresenta uma estratégia que inverte a idéia tradicional de armazenamento de dados que passa a ser baseada em colunas. De acordo com esta proposta, os dados sucessivos de cada atributo de uma tupla são armazenados em colunas separadas de forma que os valores de um mesmo atributo são armazenados consecutivamente em disco. Com esta arquitetura, é necessário trabalhar somente os valores das colunas requisitadas por uma operação de leitura ou escrita, evitando o acesso aos atributos irrelevantes para uma determinada operação definida.

O armazenamento de dados baseado em colunas já foi abordado anteriormente por outros trabalhos como em Sybase IQ [40] , Addamark [1], Bubba [7], MonetDB [5] e KDB [16]. Porém, o trabalho mais recentemente desenvolvido que aborda este assunto é o C-Store [37]. De acordo com [37], a arquitetura C-Store apresenta uma estrutura composta pelos componentes *Writeable Store* (WS), *Read-optimized Store* (RS) e *Tuple Mover*. O componente WS é responsável por disponibilizar um alto desempenho em operações de inserções e atualizações, o componente RS é definido para suportar leitura de grande quantidade de dados visando otimizar o processo de leitura. O *Tuple Mover* é responsável por movimentar os registros do WS para o RS o que possibilita disponibilizar as vantagens oferecidas pelos dois primeiros componentes. C-Store apresenta as seguintes características:

- Uma arquitetura híbrida com os componentes WS para otimizar as inserções e atualizações e o RS para otimizar a execução de consultas;
- Armazenamento redundante dos elementos de uma tabela em diversas sobreposições de projeções em ordens diferentes, de modo que uma consulta possa ser resolvida usando a projeção mais vantajosa;
- Armazenamento de colunas comprimidas utilizando técnicas de compressão;
- Código otimizado para trabalhar com armazenamento de dados orientado a colunas;
- Alto desempenho com o uso do método *K-safety* que trabalha com as sobreposições de diferentes projeções;

- Permite realizar transações com os dados orientados a colunas e utiliza *snapshop isolation* para evitar o 2PC (*Two Phase Commit*) e o bloqueio de consultas.

A arquitetura definida em C-Store apresenta um conjunto de inovações quando comparada aos demais trabalhos citadas acima por reunir uma combinação de técnicas que resultam em uma melhor performance nas operações de leitura e escrita, melhor resultado na compressão de dados e no gerenciamento de transações utilizando *snapshop isolation* e recuperação de dados. Esta proposta apresenta várias vantagens, entretanto, ela não foi definida considerando as características restritivas dos ambientes móveis como os dispositivos do tipo PDA.

A arquitetura PDM proposta neste trabalho tem como diferencial, além da estrutura definida com técnicas de armazenamento baseado em colunas, gerenciamento e compressão dos dados, o fato de ter sido projetada considerando as limitações de recursos computacionais existentes nos dispositivos portáteis do tipo PDA. Vários aspectos importantes relacionados a estes equipamentos foram considerados (limitado poder de processamento, limitada capacidade de armazenamento de dados, restrições de energia e suporte à mobilidade) o que possibilitou que a proposta definida neste trabalho atingisse benefícios bastante significativos nesta área.

Considerando-se como premissas todas as condições e restrições de *hardware* e *software* existentes nos dispositivos móveis do tipo PDA, e todas as deficiências existentes nos ambientes operacionais de *handhelds* e *palmtops* como, por exemplo, a necessidade de manipular os dados através do sistema de arquivo convencional, obteve-se os seguintes resultados nos aspectos em que a arquitetura PDM se propôs oferecer:

- Definição de estruturas que oferecem uma camada de abstração dos dados armazenados em arquivos associada a uma visão relacional dos mesmos permitindo a utilização de estruturas como banco de dados, tabelas e tuplas;
- Armazenamento de dados com maior facilidade de acesso para operações de projeção;
- Facilidade na criação e manipulação dos esquemas de dados;
- Definição e manutenção de integridade referencial;

Associado a estas características, a estrutura de armazenamento proposta neste trabalho oferece melhores condições para otimizar a aplicação das técnicas de compressão de dados, potencializando as taxas de compressão atingidas e aumentando a capacidade de armazenamento de dados nestes dispositivos.

Dessa forma, percebe-se que a idéia de projetar o armazenamento de dados baseado em colunas já foi apresentada em várias propostas [1][5][7][16][40][37] e possui algumas vantagens quando comparadas ao armazenamento de dados convencional baseado em tupla. De acordo com a proposta de gerenciamento escolhida para armazenamento baseado em colunas, estas vantagens tornam-se ainda mais evidentes. Este é o caso da arquitetura PDM que aplicada em sistemas computacionais móveis com recursos limitados, disponibiliza uma estrutura com um conjunto de benefícios pouco encontrados nestes ambientes.

Capítulo 5

Estudo de Caso: A Ferramenta *Palm Database Manager*

Este capítulo apresenta a ferramenta PDM (Palm Database Manager) como um estudo de caso para a estratégia de armazenamento de dados proposta no Capítulo 4.

5.1 Introdução

No Capítulo 3 foram apresentadas algumas ferramentas de gerenciamento de bancos de dados para ambientes móveis. Tais ferramentas têm como características comuns uma arquitetura cliente-servidor em que os dispositivos móveis possuem uma versão cliente do sistema gerenciador de bancos de dados móveis. A partir da versão cliente é possível acessar os dados em uma cópia local no dispositivo móvel ou em um servidor de banco de dados de uma estação fixa através de acesso remoto. As ferramentas de gerenciamento de banco de dados apresentadas utilizam o padrão de armazenamento convencional em que uma tupla possui dados de vários atributos da tabela. No Capítulo 4, foi apresentada uma estrutura diferenciada de armazenamento dos dados, em que uma tupla possui os dados de um mesmo atributo da tabela com o objetivo de disponibilizar uma visão relacional para facilitar acesso e gerenciamento dos dados. Para disponibilizar esta nova estrutura, foi criada a biblioteca DBLib que possui um conjunto de funcionalidades que permitem trabalhar com a estrutura definida, além de oferecer propriedades essenciais dos bancos de dados relacionais.

Estas são importantes funcionalidades para serem disponibilizadas em dispositivos do tipo PDA. Isto porque no ambiente operacional Palm OS, por exemplo, o desenvolvedor é inteiramente responsável por determinar como os dados são gravados nos arquivos. A persistência dos dados não acontece de forma transparente para a aplicação cliente. Para existir restrições de integridade entre os dados, por exemplo, a aplicação deve garantir esse tipo de propriedade.

A motivação para o desenvolvimento da ferramenta PDM está na necessidade de validar e avaliar a estrutura definida no Capítulo 4, bem como disponibilizar para o usuário uma aplicação capaz de trabalhar com uma visão relacional dos dados oferecendo um maior grau de usabilidade e flexibilidade em relação às estruturas de armazenamento de dados existentes.

O desenvolvimento da aplicação PDM permite avaliar tanto a estrutura de armazenamento proposta como a utilização das funções definidas na biblioteca DBLib. É importante destacar que esta abordagem pode ser implementada em qualquer ambiente operacional desenvolvido para PDAs como, por exemplo, Palm OS ou Pocket PC. Contudo, para demonstrar a eficiência do mecanismo proposto, foi utilizado o sistema operacional Palm OS como um estudo de caso nos experimentos realizados.

5.2 A Ferramenta PDM

A ferramenta *Palm Database Manager* (PDM) [32], desenvolvida em C++, tem como objetivo fornecer propriedades de um sistema gerenciador de banco de dados para ambientes móveis com sistema operacional Palm OS.

A ferramenta PDM foi desenvolvida a partir da biblioteca DBLib que disponibiliza um conjunto de funções com o objetivo de prover uma camada de abstração entre aplicação e arquivos do ambiente Palm OS, visando abstrair o conceito de arquivo. Para tanto esta biblioteca utiliza as funções da API da Palm: *Data Manager*. Com isto, qualquer aplicação pode trabalhar com a idéia de banco de dados relacionais e não mais com arquivos (PDB, por exemplo).

A DBLib oferece um conjunto de funções para o acesso aos dados abstraindo-se a maneira como as funções da *Data Manager* realizam a manipulação dos dados em memória. Ela apresenta uma visão lógica sobre os dados, não disponibilizando a forma como os dados estão armazenados fisicamente, o que facilita a manipulação destes através das aplicações. Com isto, um desenvolvedor de aplicação para o ambiente operacional Palm OS, por exemplo, pode manipular os bancos de dados com suas respectivas tabelas e restrições de integridade como se estive trabalhando com um banco de dados relacional tradicional existente em computadores *desktop*.

5.2.1 Estrutura da Arquitetura PDM

A Figura 38 apresenta a estrutura inicial da arquitetura sobre a qual a ferramenta PDM foi construída. A arquitetura é composta por quatro camadas: apresentação, representação lógica, APIs da Palm e camada de dados. Cada camada é composta por um ou mais componentes. A seguir será descrita cada camada da arquitetura, bem como os possíveis componentes de cada camada. Entre os componentes da arquitetura estão aqueles relacionados à compactação dos dados armazenados em memória (CompLib) e à comunicação sem fio que disponibiliza o acesso aos dados de forma local ou remota (BDConnLib e ConnLib) que serão inseridos como componentes desta arquitetura mais adiante.

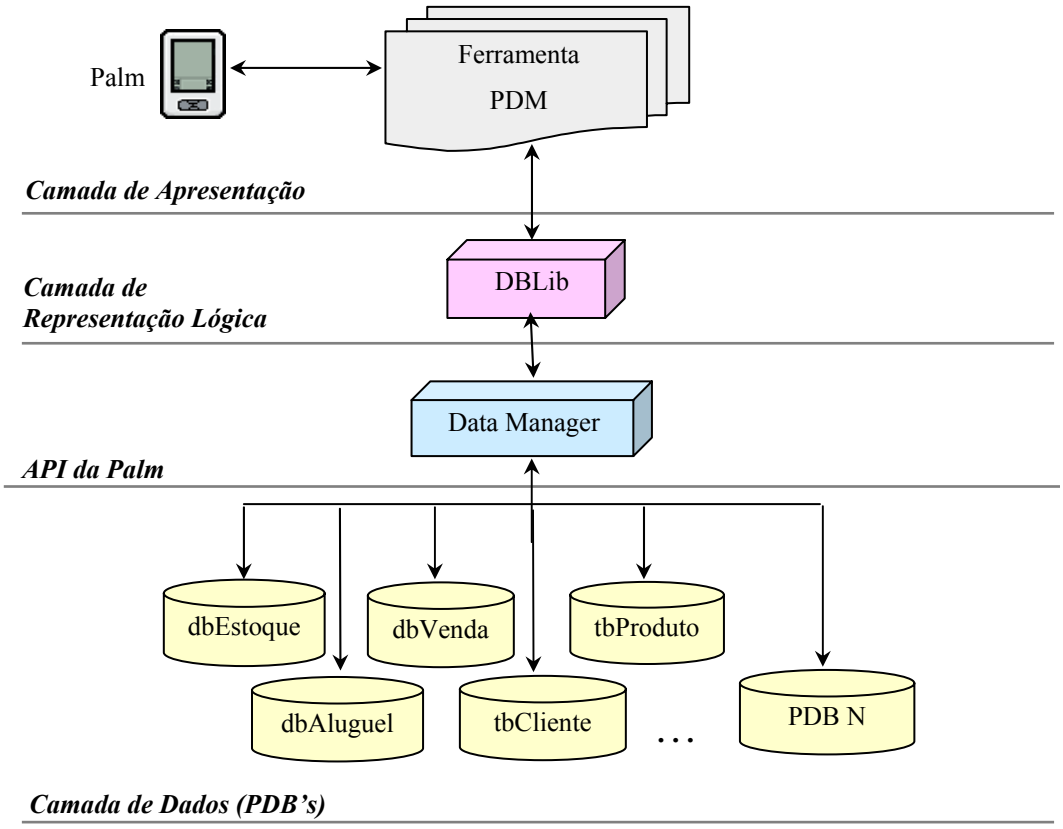

Figura 38 - Visão Geral da Arquitetura PDM (Parte I)

A primeira camada é composta pela ferramenta PDM e disponibiliza uma *interface* gráfica que permite ao usuário acessar os dados a partir das funções da DBLib. Essa aplicação tem a finalidade de gerenciar e manipular as informações armazenadas nos bancos de dados, de forma que os dados poderão ser acessados através de uma visão relacional.

A segunda camada, camada de representação lógica, é composta inicialmente pelas funções da biblioteca DBLib (mais adiante serão acrescentadas novas bibliotecas nesta camada). A DBLib define uma arquitetura de armazenamento de dados que utiliza o conceito de metadados semelhante ao conceito de esquema dos bancos de dados relacionais, possibilitando a definição de banco de dados, tabelas, colunas e restrições de integridade. Essas funções permitem ao usuário criar, alterar e excluir todas essas estruturas, além de possibilitar a definição de chave primária e chave estrangeira.

Todas as funcionalidades da DBLib estão disponibilizadas pela ferramenta PDM a partir de *interfaces* gráficas. Ao contrário do acesso via programação oferecido pela API *Data Manager*, as funções da DBLib oferecem uma melhor interação do usuário com os esquemas de dados e um maior grau de usabilidade das funções, uma vez que elas podem ser acessadas via *interfaces* gráficas. Isto também oferece uma maior flexibilidade na definição das estruturas de armazenamento como tabelas e colunas. As funções disponibilizadas pela DBLib permitem manipular tais estruturas oferecendo várias funcionalidades conforme relação apresentada no Apêndice A.

Dessa forma, através da ferramenta PDM, o usuário pode manipular todas as estruturas do banco de dados definidas a partir das funções disponibilizadas pela DBLib. Além da manipulação dos esquemas dos bancos de dados, a ferramenta permite visualizar os dados existentes nas tabelas. Para realizar a manipulação dos dados das tabelas, a DBLib possui funções que realizam operações de inclusão, exclusão, alteração e busca dos dados armazenados em uma determinada estrutura.

A terceira camada é composta pelas APIs da Palm que são responsáveis por acessar os dados diretamente em memória através de uma visão física dos mesmos. A DBLib acessa a *Data Manager* para efetivar suas operações em memória e realizar todos os acessos aos dados solicitados através da visão lógica que oferece.

A camada de dados é o último nível da arquitetura, sendo formada por todos os dados armazenados no sistema de arquivos do Palm OS e que são gerados pela aplicação PDM a partir da biblioteca DBLib. Conforme a estrutura definida, a camada de dados é acessada apenas pela API *Data Manager*.

Para que a arquitetura PDM atendesse as necessidades de otimização na utilização de espaço em memória dos dispositivos portáteis, foi incluída na camada de representação

lógica a biblioteca de compactação *Compact Library* (CompLib), que é responsável pelo armazenamento e acesso aos dados de forma compactada. Com isto, a arquitetura PDM passa a ser composta pelos componentes apresentados na Figura 39.

Conforme a estrutura de armazenamento de dados definida no Capítulo 4, cada tupla da tabela armazena os dados de uma mesma coluna ou atributo da tabela de forma que tais dados são necessariamente sempre do mesmo tipo. Essa característica permite aplicar a melhor técnica de compressão para cada tupla da tabela com o objetivo de atingir a maior taxa de compressão possível para cada atributo da tabela.

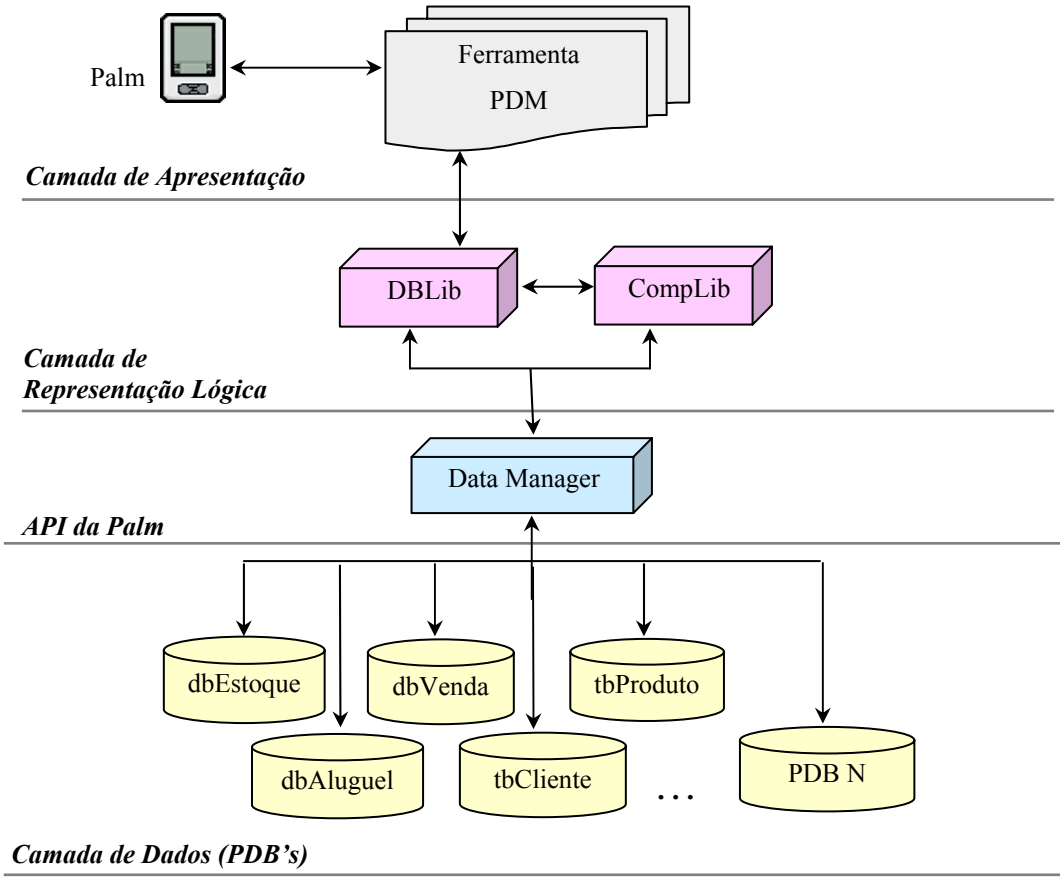

Figura 39 - Visão Geral da Arquitetura PDM (Parte II)

A biblioteca CompLib é responsável pelo armazenamento e o acesso aos dados comprimidos contidos nos arquivos. Esta biblioteca utiliza as técnicas de compressão que foram definidas em [4] e citadas na seção 4.4 deste trabalho. Tais técnicas possibilitam o armazenamento dos dados em formato de compressão e a aplicação da estratégia de descompressão mais adequada para o acesso aos dados comprimidos de acordo com o tipo de dado.

Dessa forma, a aplicação PDM acessa as funções da DBLib que, por sua vez, acessam as funções da CompLib para armazenar os dados em formato comprimido ou para descomprimir os dados recuperados. O gerenciamento dos registros de dados é realizado apenas pela a CompLib ficando totalmente transparente para a DBLib e para a aplicação PDM a forma como esses dados estão armazenados e recuperados.

Para possibilitar este grau de transparência da estrutura proposta, foi necessária a inclusão de novas estruturas de controle nesta arquitetura de armazenamento que possibilitam melhor gerenciar os dados em formato de compressão. Esta estratégia de compressão de dados que considera novas estruturas de controle permite que a CompLib associada à DBLib ofereçam benefícios como: (1) otimização de espaço em memória e (2) maior padronização no armazenamento facilitando o controle da consistência dos dados utilizados por uma determinada aplicação.

5.2.1.1 Comunicação

Para atender a uma necessidade essencial dos dispositivos portáteis do tipo PDA, foram adicionadas à arquitetura PDM funcionalidades relacionadas à mobilidade. A possibilidade de realizar comunicação remota entre dispositivos móveis é uma característica importante a ser considerada uma vez que este é um dos principais princípios da computação móvel.

Dessa forma, na camada de representação lógica foram adicionadas as funções da biblioteca *Connect Library* (ConnLib). A ConnLib oferece o suporte necessário para que a aplicação possa estabelecer uma conexão remota com um determinado banco de dados existente em outro dispositivo móvel, de acordo com os tipos de conexões disponíveis no equipamento. Para realizar esse acesso remoto, a biblioteca ConnLib precisa utilizar as funcionalidades das APIs *Net Library* (NetLib) e *Bluetooth Library* (BtLib) fornecidas pela Palm, conforme ilustra a Figura 40.

Entretanto, para dar suporte a funcionalidade de comunicação remota foi necessária a inclusão da camada de integração definida pela *Database Connect Library* (DBConnLib). Esta camada permite realizar a integração entre a aplicação PDM e as APIs DBLib e ConnLib e tem como objetivo permitir que a aplicação realize o acesso às funções das bibliotecas do nível abaixo, para que as ações realizadas pelo usuário possam ser interpretadas e efetivadas no banco de dados local ou remoto.

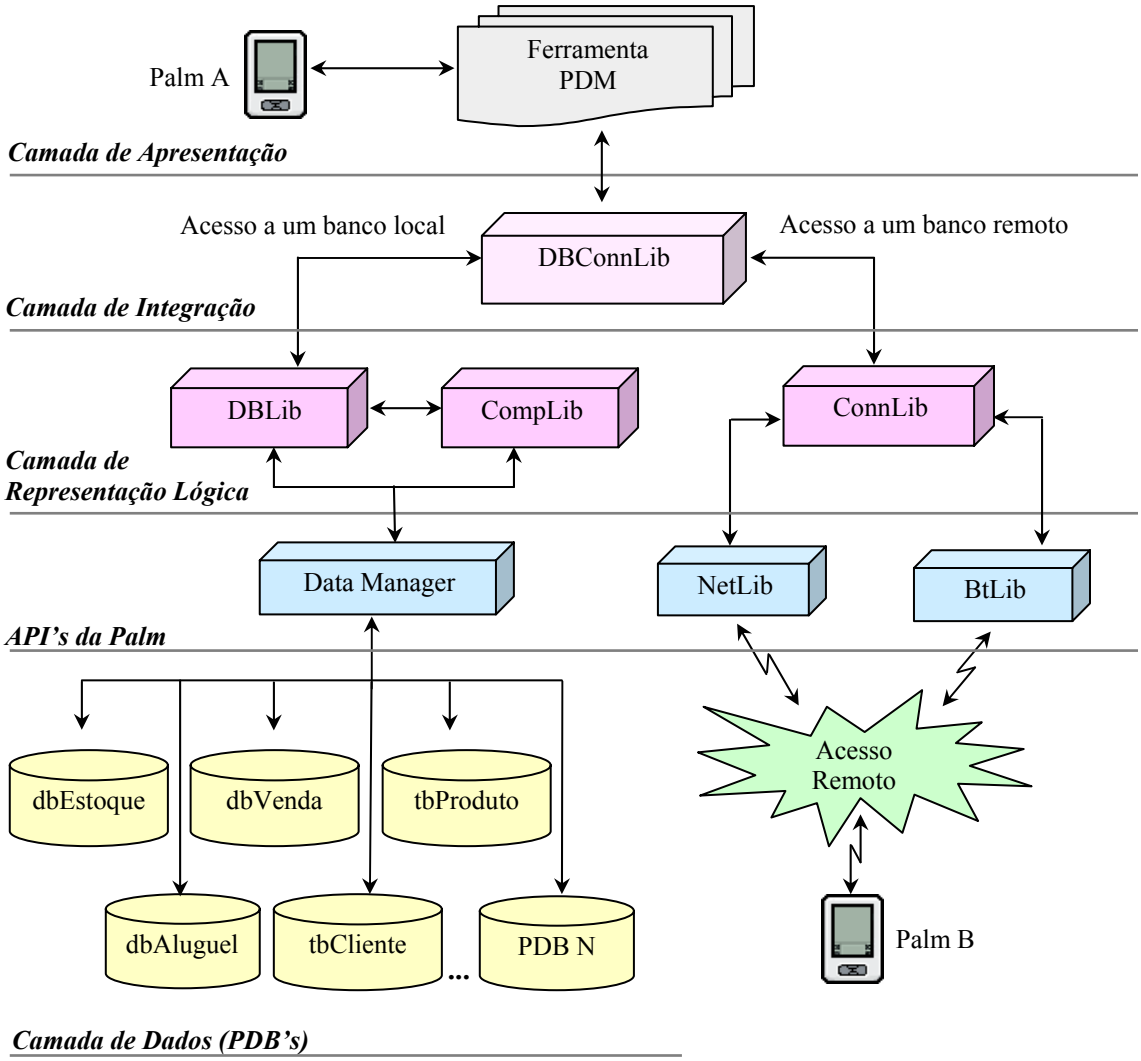

Figura 40 - Visão Geral da Arquitetura PDM (Parte III)

A aplicação PDM pode realizar o acesso aos dados de forma local ou remota. Se for local, a aplicação chama a DBConnLib que executa as funções da biblioteca DBLib. Se a conexão for remota, o usuário deve determinar em qual dispositivo e de que maneira (por *Bluetooth* ou *Network*) a conexão deve ser estabelecida. Para isso, o dispositivo a ser acessado precisa estar executando a aplicação PDM para que possa receber a solicitação remota e executá-la localmente. A cada operação realizada no banco de dados remoto, a aplicação chama a DBConnLib que executa a função de envio de mensagem da ConnLib. Quando essa mensagem for recebida no dispositivo remoto, a aplicação chama a DBConnLib que, por sua vez, interpreta a mensagem recebida formatando-a no padrão esperado pela DBLib. Esta executa a operação localmente e envia para a DBConnLib a resposta apropriada que, por sua vez, a retorna ao dispositivo solicitante. Assim, o usuário pode realizar, em um banco de

dados localizado em outro dispositivo portátil, todas as operações de definição e manipulação de bancos de dados disponibilizadas pela biblioteca DBLib remotamente. A DBLib sempre executa suas funções localmente, mesmo que a solicitação seja de um cliente remoto. A DBConnLib é a responsável por gerenciar o acesso local ou remoto realizado pela aplicação.

Uma característica importante da arquitetura PDM é a independência existente entre as APIs DBLib e ConnLib, pois isto permite que ambas possam ser utilizadas isoladamente por outras aplicações.

A biblioteca ConnLib disponibiliza um conjunto de funções que permitem o estabelecimento de conexões com dispositivos remotos e a troca de mensagens entre eles, de forma que nela estão todas as funcionalidades referentes ao acesso remoto. Essa biblioteca utiliza as funcionalidades das APIs de conexão do ambiente Palm OS: NetLib e BtLib. A primeira oferece serviços básicos de rede usando TCP (*Transmission Control Protocol*) e UDP (*User Datagram Protocol*) através de *sockets*. De acordo com as funcionalidades suportadas pelo aparelho utilizado, o usuário pode determinar se deseja conectar-se através de Wi-Fi ou utilizando o serviço GPRS (*General Packet Radio Service*). A BtLib também oferece a possibilidade de acesso a outros dispositivos. Inicialmente, o usuário deve utilizar a função de descobrir outros dispositivos ativos ao alcance do sinal e, em seguida, pode escolher o dispositivo com o qual deseja se conectar.

Após conectar-se ao dispositivo escolhido, o usuário poderá abstrair-se de qual tipo de conexão está utilizando. Todas as funcionalidades disponibilizadas pela ferramenta PDM serão oferecidas de forma idêntica para os diversos tipos de serviços. Depois de definido o tipo de acesso, o usuário pode manipular as estruturas do banco de dados remoto normalmente.

Dessa forma, a ferramenta PDM apresenta funcionalidades que oferecem um melhor gerenciamento dos dados armazenados em um dispositivo portátil do tipo PDA (*palmtops* e *handhelds*) e disponibiliza a conectividade a bancos de dados remotos a partir da arquitetura proposta.

5.2.2 Principais Funcionalidades da Ferramenta PDM

A ferramenta PDM disponibiliza uma *interface* gráfica para o usuário com o objetivo de permitir que estruturas de dados sejam criadas seguindo a arquitetura PDM. A

seguir são apresentadas algumas funcionalidades existentes na ferramenta que atendem a este propósito. A aplicação disponibiliza uma opção para o usuário criar um banco de dados seguindo a estrutura definida no Capítulo 4. A Figura 41 ilustra os bancos de dados criados a partir desta estrutura utilizando a função DBLibCreateDatabase.

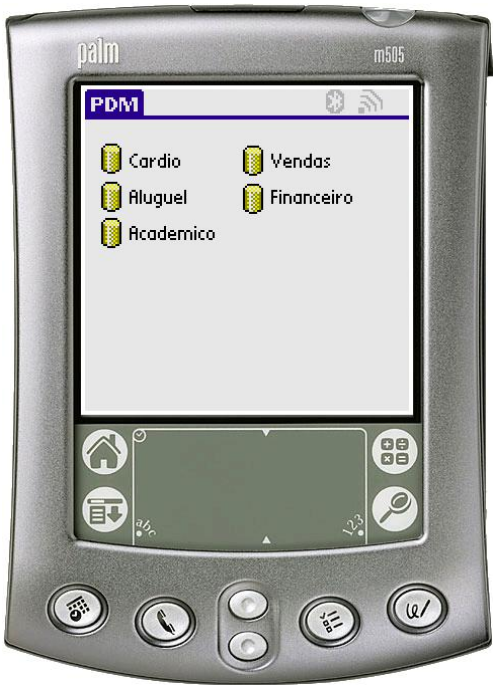

Figura 41 - Tela Inicial da ferramenta PDM

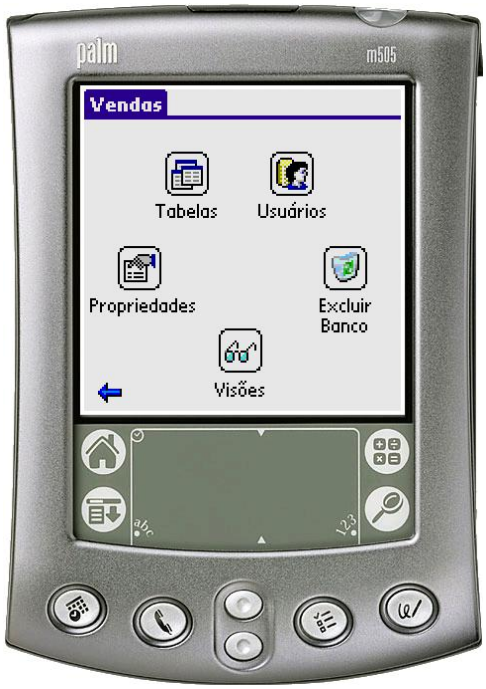

Figura 42 - Componentes do banco de dados

Depois de criado o banco de dados é possível acrescentar os elementos essenciais a sua estrutura como essa tais como as tabelas, colunas, relacionamentos entre tabelas, visões e usuários. Para acessar os componentes de um banco de dados basta selecionar o ícone do banco desejado na tela inicial. Após selecionar o banco de dados desejado, o nome do banco selecionado será mostrado no título da tela. O usuário poderá acessar qualquer uma das opções disponíveis para manipular o banco de dados, conforme ilustra a Figura 42.

A ferramenta também disponibiliza informações relacionadas às propriedades do banco de dados. Nela são apresentadas informações gerais sobre o banco de dados selecionado como: data da criação, data da última alteração, número de tabelas, número de visões, número de relacionamentos, número de usuários e o tamanho total do banco de dados.

5.2.2.1 Tabelas

A opção Tabelas disponibiliza todas as tabelas existentes no banco de dados selecionado, conforme mostra a Figura 43. No canto superior direito da tela será mostrado um ícone com o nome do banco selecionado anteriormente.

Figura 43 - Tabelas do banco de dados

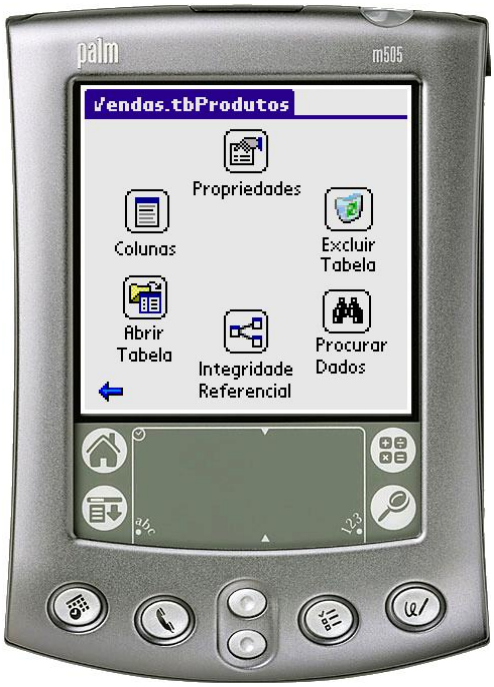

Figura 44 – Componentes da tabela

É possível criar novas tabelas no banco de selecionado. Para se criar uma tabela, será necessário informar o nome da tabela e suas colunas. Ao definir as colunas da tabela será necessário informar o nome, o tipo e, se necessário, o tamanho da coluna ou a quantidade de casas decimais. Além disso, é possível definir se a coluna é uma chave primária e se ela pode possuir valores nulos. Caso a coluna não possa ter valores nulos e não seja definida como chave primária, é habilitado um campo que permite o usuário informar um valor padrão para a coluna que está sendo criada. Assim, esse valor padrão será inserido sempre que o usuário tentar incluir uma tupla com este campo vazio. Depois de informada as propriedades da nova coluna, esta pode ser associada à tabela a ser criada. Para criar uma tabela é utilizada a função DBLibCreateTable. A tabela criada possui alguns dos aspectos que podem ser trabalhados pelo usuário como: Colunas, Propriedades, Integridade Referencial, Excluir Tabela, Abrir Tabela e Procurar Dados, conforme ilustra a Figura 44.

A opção Propriedades existente na tela de componentes da tabela (ver Figura 44) apresenta as informações gerais da tabela selecionada como: data de criação, data da última alteração, número de colunas, número de registros, número de relacionamentos e tamanho da tabela. A opção Excluir Tabela irá excluir a tabela em questão. Neste caso, todos os dados relacionados a esta tabela serão excluídos juntamente com a tabela.

5.2.2.1.1 Colunas

É possível acessar todos os atributos da tabela através do ícone Colunas, conforme mostra a Figura 45. Também é possível acessar as propriedades definidas para cada atributo da tabela. Para tanto, basta clicar sobre uma coluna específica da tabela para que suas propriedades possam ser visualizadas.

Depois de criada uma tabela, é possível adicionar novas colunas através da tela que lista as colunas da tabela (ver Figura 45). Se a coluna adicionada possuir valor padrão definido, ao adicionar a coluna esse valor padrão será inserido nos registros já existentes da tabela. O valor padrão só é considerado quando o usuário não informar dado para o referido campo. Assim, caso o usuário, ao inserir uma nova tupla, não informar um valor para a coluna com valor padrão, a função de inserção irá considerar o valor padrão definido para a coluna na tupla a ser inserida. Para adicionar uma nova coluna foi implementada a função DBLibAddColumn.

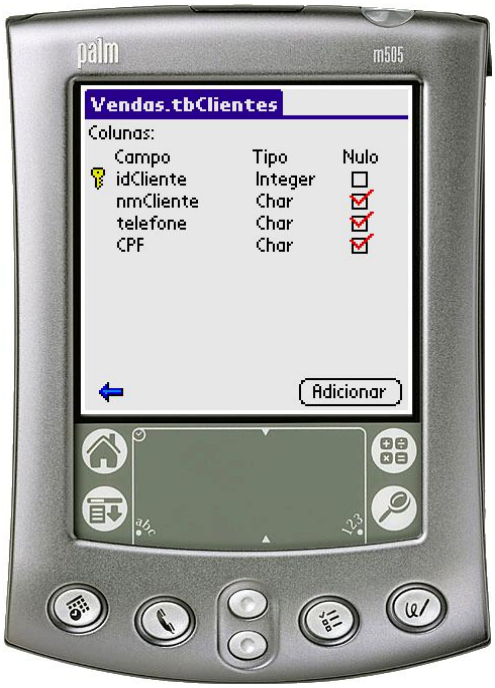

Figura 45 - Lista das colunas existentes em uma tabela

5.2.2.1.2 Integridade Referencial

A ferramenta também implementa a consistência dos dados em relação às colunas definidas como chave primária e chave estrangeira. As restrições de integridade referencial são definidas na tela mostrada na Figura 46.

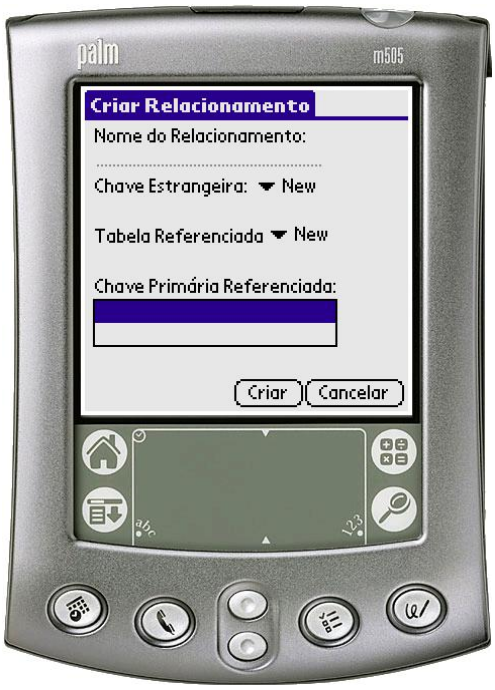

Figura 46 – Criação de um novo relacionamento entre tabelas (Integridade Referencial)

As colunas definidas como chave primária não poderão ter valores repetidos. Já as colunas definidas como chave estrangeira terão seus valores referenciados pelos campos definidos na integridade referencial. Na tela de criação de uma integridade referencial será necessário informar o nome do relacionamento, a coluna da tabela atual que será definida como chave estrangeira, o nome da tabela referenciada e sua respectiva coluna referenciada (chave primária). Para criar relacionamentos entre tabelas foi implementada a função DBLibCreateRelationship e para manter a consistência dos dados de acordo com as restrições de integridade, como chave primária (integridade de chave) ou chave estrangeira (integridade referencial) definidas pelo usuário, foram implementadas as funções:

- DBLibCheckColumnPrimaryKey;
- DBLibCheckColumnForeignKey;
- DBLibCheckRelationship;
- DBLibCheckRelationshipReferences.

5.2.2.1.3 Consulta

A ferramenta disponibiliza uma opção para consultar os dados a partir de filtros utilizando a estratégia QBE (*Query By Exemple*), conforme ilustra a Figura 47.

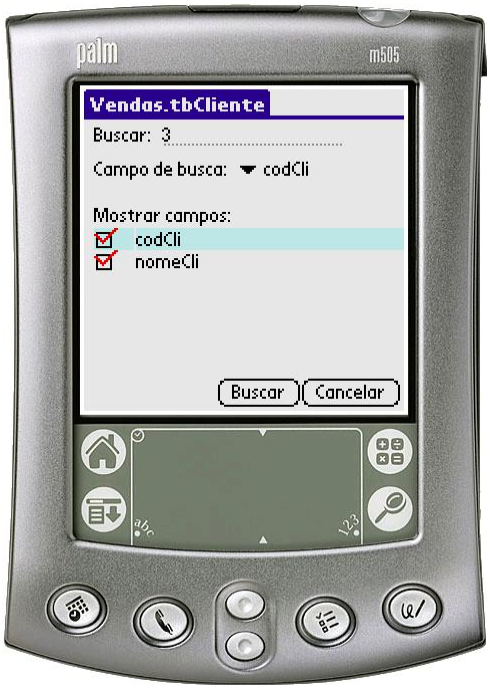

Figura 47 – Consulta de dados em uma tabela

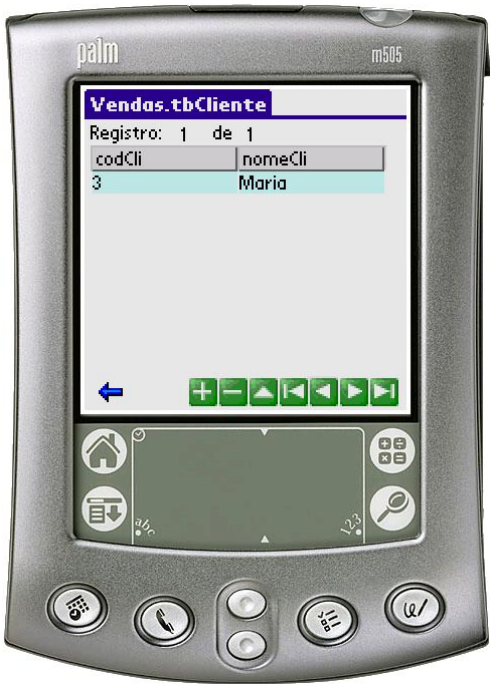

Figura 48 - Resultado de uma operação de busca

Para definir uma consulta é necessário informar o valor de busca, a coluna que se deseja realizar a busca e os campos que devem ser mostrados no resultado da busca. Após realizar a busca será disponibilizado o resultado da consulta com os dados que atendem as condições de busca definida pelo usuário, conforme mostra a Figura 48. Para realizar consultas sobre os dados armazenados nas tabelas foram definidas as funções DBLibFindRecord e DBLibGetRecord.

5.3 Resultados Experimentais

A ferramenta *Palm Database Manager* (PDM) [32], foi desenvolvida com o objetivo de avaliar e validar as estruturas de armazenamento propostas neste trabalho que corresponde ao conceito diferenciado de gravar os mesmo tipos de dados (colunas) em um mesmo registro. Associado a isto a ferramenta realiza a compressão desses dados visando, otimizar a utilização de espaço em memória. Esta ferramenta fornece suporte para que dados

armazenados em ambientes Palm OS sejam vistos e manipulados como dados de bancos de dados relacionais em formato de compressão.

Para avaliar a estratégia de armazenamento apresentada neste trabalho, foram realizados alguns experimentos sobre dois tipos de bancos de dados baseados na estrutura de armazenamento proposta: (1) bancos de dados não-comprimidos e (2) bancos de dados comprimidos. Como a estrutura de armazenamento definida apresenta melhoras relacionadas à manipulação dos dados disponibilizados através de uma visão relacional, optou-se por realizar experimentos avaliando possíveis benefícios relacionados à otimização do armazenamento quando a estrutura definida pela DBLib é associada às técnicas de compressão definidas pela CompLib.

Conforme já mencionado, as estratégias de compressão de dados apresentadas anteriormente na seção 4.4 são uma extensão da estratégia de armazenamento de dados baseados em colunas proposta neste trabalho. Dessa forma, o objetivo dos testes realizados é mensurar os benefícios que podem ser obtidos utilizando a estrutura de armazenamento dados proposta baseada em colunas associando-se a ela as técnicas de compressão definidas.

Os resultados obtidos levaram em consideração os tempos de resposta e as taxas de compressão obtidas sobre as tabelas. Foram utilizadas tabelas com tipos de dados bastante heterogêneos e de fontes distintas, possibilitando uma melhor avaliação dos resultados. Os experimentos foram realizados no *handheld Tungsten C* da Palm com 64 MB de memória e processador Intel PXA255 de 400 MHz.

A Tabela 8 mostra os tamanhos (em *bytes*) das tabelas comprimidas e não comprimidas associados às taxas de compressão atingida para cada experimento. A Tabela 9 apresenta os tempos de resposta para *bulkload* de cada uma das tabelas.

Tabela	Não-comprimida	Comprimida	Taxa de Compressão
ZonaEleitorais	49.000	23.162	52,73%
InfoMunicipio	60.000	24.962	58,40%
CursosCapes	84.360	42.835	49,22%
ConceitoPorCentro	45.000	28.783	36,04%
Total	*238.360*	*119.742*	*49,76%*

Tabela 8 - Tamanho (em *bytes*) de Tabelas Comprimidas e Não-Comprimidas

As melhores taxas de compressão foram obtidas nas tabelas `ZonaEleitorais` e `InfoMunicipio` porque estas possuem muitos valores numéricos. A predominância de colunas do tipo Inteiro resultou em uma alta taxa de compressão porque a grande maioria dos valores precisou de apenas um ou dois *bytes* para seu armazenamento, reduzindo bastante o espaço em relação à tabela original que necessitava de quatro *bytes* para cada valor dessas colunas. Dessa forma, de acordo com a predominância de tipos de dados na tabela é possível atingir taxas de compressão mais altas quando comparados a tabelas com tipos de dados mais heterogêneos.

No entanto, essas duas tabelas apresentaram os maiores aumentos de tempo de execução do *bulkload* em relação à versão não-comprimida. O aumento na tabela `InfoMunicipio` foi de quase 30% e para a tabela `ZonaEleitorais` foi de 46,75%. O motivo foi o mesmo que elevou a taxa de compressão: muitas colunas do tipo Inteiro. Para este tipo de dado, a necessidade de se calcular o tamanho e o *offset* de cada valor é bastante cara do ponto de vista computacional porque estas informações não estão em nível de *byte*, mas sim em nível de "par de *bits*".

Tabela	Não-comprimida	Comprimida	Tempo de *Bulkload*
ZonaEleitorais	5.070	7.440	46,75%
InfoMunicipio	7.430	9.580	28,94%
CursosCapes	3.990	4.730	18,55%
ConceitoPorCentro	6.810	7.510	10,28%
Total	*23.300*	*29.260*	*25,56%*

Tabela 9 - Tempo (em ms) de Tabelas Comprimidas e Não-Comprimidas

A tabela `CursosCapes` apresentava muitas colunas do tipo `CHAR` o que diminuiu um pouco a taxa de compressão (49,22%) porque a grande maioria dos valores destas colunas tinha tamanhos muito próximos dos declarados na criação da tabela. No entanto, o tempo de resposta aumentou apenas 18% porque para valores do tipo `CHAR` o tamanho está armazenado sempre em 1 *byte* e não há necessidade de se calcular o *offset.*

A tabela `ConceitoPorCentro` apresentou a mais baixa taxa de compressão porque apesar de existirem algumas colunas do tipo Booleano (cuja taxa de compressão é de quase 90%) a maior parte do seu tamanho deve-se a uma coluna do tipo `CHAR` com tamanhos muito

próximos ao máximo da coluna. Esta predominância de *strings* foi o motivo do menor aumento de tempo obtido: 10 %.

A taxa de compressão total e os tempos de resposta obtidos são melhores do que em outras estratégias semelhantes aplicadas sobre bancos de dados como em [45], onde as taxas de compressão variaram entre 10% e 45 % (contra taxas de 36% a 52% encontradas nesta proposta) e os tempos de *bulkload* estiveram de 20% a 50% maiores nos bancos de dados comprimidos (contra aumentos de 10% a 46 % encontrados nesta proposta). Os resultados mostraram também que quanto maior a quantidade de colunas do tipo Inteiro, maior será a taxa de compressão e maior será o aumento nos tempos de resposta. Por outro lado, se houver muitos valores do tipo `CHAR` na tabela, haverá uma menor taxa de compressão e um menor aumento no tempo de resposta das tabelas.

A Figura 49 apresenta um gráfico com a variação do tamanho de acordo com o aumento do número de tuplas das tabelas tanto para a versão comprimida quanto para a versão original. É possível perceber que a taxa de compressão se mantém constante em relação ao aumento do número de tuplas.

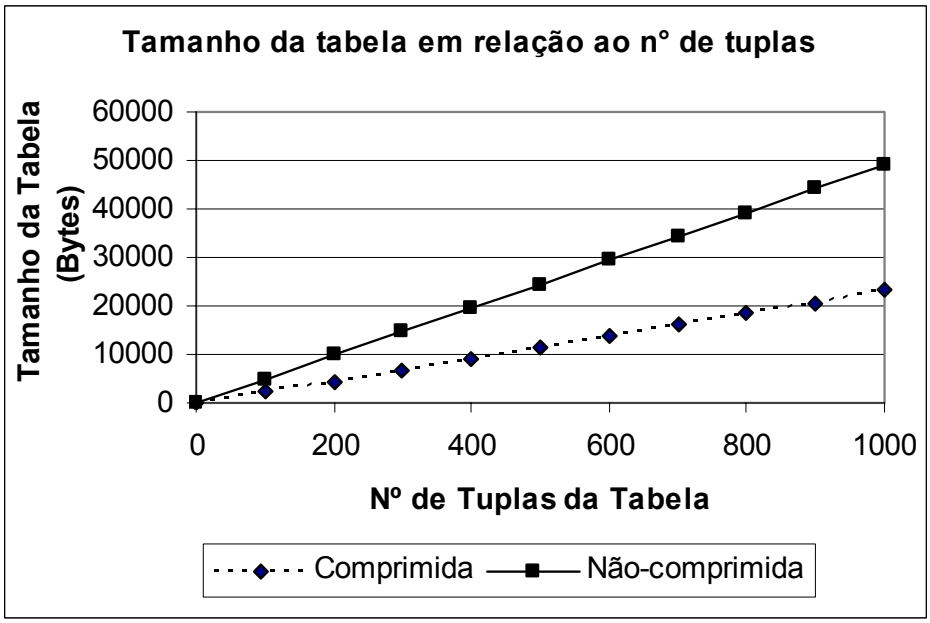

Figura 49 - Variação entre a quantidade de tuplas e o tamanho da tabela (comprimida e não-comprimida)

A Figura 50 mostra um gráfico com o crescimento do tamanho da tabela `ZonaEleitorais` comprimida de acordo com o aumento do tamanho dessa tabela não-comprimida. Pode-se perceber que à medida que a tabela não-comprimida aumenta de

tamanho, a taxa de crescimento do tamanho da tabela comprimida mantém-se constante, o que indica que as taxas de compressão obtidas serão, aproximadamente, as mesmas para tabelas com grande quantidade de dados.

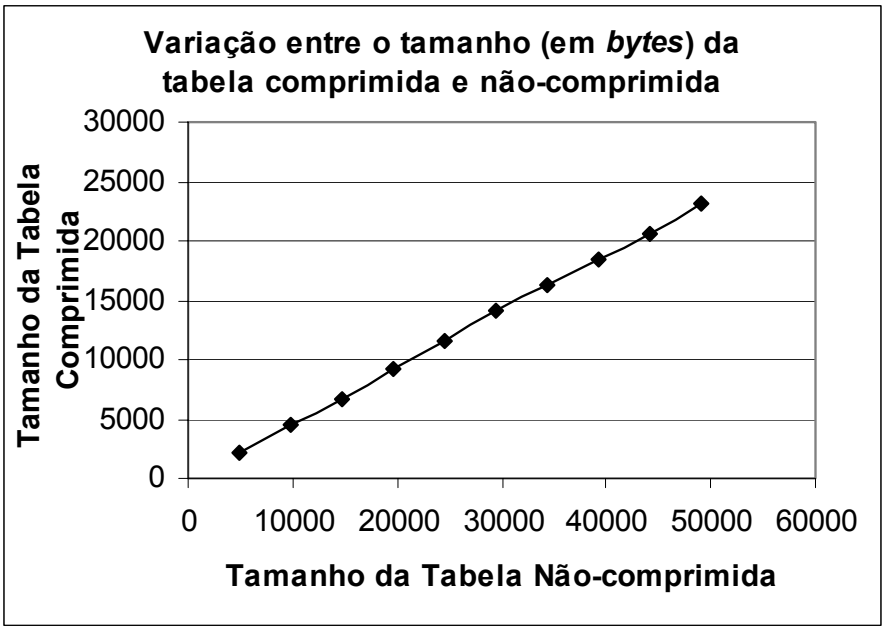

Figura 50 - Relação entre o aumento do tamanho da tabela ZonaEleitorais

É importante ressaltar também que todos os testes foram realizados em um dispositivo de alto poder de processamento: o *Tungsten C*. Em experimentos realizados em dispositivos menos potentes, os aumentos nos tempos de resposta foram consideravelmente menores porque nestes dispositivos a maior parte do tempo de processamento estará voltado para outras operações.

Dessa forma, é possível concluir que para se atingir resultados positivos em relação a taxas de compressão e tempo de acesso aos dados quando comparados com outras estratégias semelhantes aplicadas sobre bancos de dados com estruturas de armazenamento tradicionais como em [45], a estrutura de armazenamento proposta neste trabalho contribuiu de maneira essencial para que tais resultados fossem obtidos. A forma de armazenamento em que cada registro da tabela armazena dados de um mesmo tipo, permitiu que a aplicação das técnicas de compressão definidas baseadas na arquitetura proposta neste trabalho atingissem taxas superiores a outras estratégias de compressão aplicadas em dados armazenados seguindo a estrutura de armazenamento convencional.

Capítulo 6

Conclusão

Este capítulo apresenta as conclusões sobre a estratégia de armazenamento de dados proposta nesta dissertação e os trabalhos futuros sobre o assunto.

6.1 Considerações Finais e Contribuições do Trabalho

A popularização dos dispositivos móvel tem propiciado um crescimento na área de computação móvel motivando a produção de aplicações voltadas para equipamentos portáteis do tipo PDAs. Esta crescente demanda tem ocasionado o surgimento de várias linhas de pesquisa no sentido de solucionar problemas inerentes a estes ambientes. Características como restrições de energia, reduzida área de armazenamento e baixo poder de processamento são grandes motivadores para a busca de soluções eficientes no sentido de otimização de armazenamento, processamento de consultas e utilização de energia.

Neste sentido, sistemas operacionais são desenvolvidos visando atender os aspectos relacionados a estes aspectos. Tais sistemas apresentam características como portabilidade, suporte à comunicação sem fio, gestão de energia e necessitam ocupar pouco espaço em memória visando otimizar a utilização de memória para o armazenamento de dados. Foram apresentadas algumas características dos sistemas operacionais Palm OS e Windows Mobile. O primeiro ambiente foi descrito mais detalhadamente por ser este o sistema operacional utilizado no estudo de caso deste trabalho.

Também foram apresentadas ferramentas que disponibilizam o suporte ao gerenciamento de bancos de dados móveis. Tais aplicações oferecem funcionalidades como sincronização, replicação, gerenciamento de transações, restrições de integridade, criptografia dos dados, segurança e acesso remoto a dados. O objetivo principal é de facilitar a comunicação realizada entre os clientes, com seus dispositivos portáteis, e uma estação fixa onde pode residir um servidor de banco de dados em ambiente *desktop* que centraliza todas as informações dos bancos de dados móveis.

Baseada nas necessidades existentes, é proposta uma estratégia que permite armazenar e gerenciar dados nestes ambientes de forma a disponibilizar de uma visão relacional sobre os dados armazenados como arquivos convencionais. A proposta deste trabalho apresenta uma estratégia de armazenamento de dados baseada em colunas em que os dados de uma mesma coluna (ou atributo) são armazenados continuamente em memória. Esta estrutura consiste em considerar que cada registro (*chunk*) armazena informações de um mesmo tipo de dado, correspondendo a uma coluna. Dessa forma, uma tupla da tabela passa a armazenar os dados de um mesmo atributo.

A principal motivação para definir a idéia da proposta apresentada neste trabalho é a necessidade de criar um mecanismo mais abrangente que permitisse melhor gerenciar os dados em dispositivos móveis do tipo PDA associado à possibilidade de flexibilizar as estruturas de armazenamento no sentido de facilitar alterações dos registros de dados existentes nos arquivos através da evolução de esquemas de dados. Além disto, tal proposta oferece os fundamentos necessários para disponibilizar funcionalidades que garantam a consistência dos dados através de restrições de integridade e potencializar a utilização de técnicas de compressão de dados visando otimizar o espaço utilizado em memória.

A estratégia de armazenamento apresentada neste trabalho, definida como arquitetura PDM, oferece os benefícios existentes em um banco de dados relacional para ambientes de recursos computacionais limitados. Esta proposta provê uma camada de abstração entre a aplicação e os dados armazenados em arquivos através das funções da DBLib. Esta biblioteca oferece a possibilidade de definir bancos de dados, tabelas, restrições de integridade (integridade de chave e integridade referencial) e visões sobre os dados de uma tabela. Estas características oferecem uma garantia da consistência dos dados e facilitam a sincronização dos dados existentes nos dispositivos PDAs para um banco de dados central existente no ambiente *desktop.* A abordagem proposta também permite que a aplicação de técnicas de compressão de dados possa ser otimizada de forma que os dados possam ser armazenados e acessados em formato de compressão, otimizando a utilização de espaço em memória e potencializando a capacidade de armazenamento de dados em tais dispositivos.

A proposta apresentada também oferece melhoria na manutenção das estruturas de armazenamento dos dados através da facilidade de alteração das estruturas de esquemas quando comparada às estruturas primitivas de armazenamento (sistema de arquivo) dos sistemas operacionais dos dispositivos do tipo PDA, como, por exemplo, o Palm OS.

Para validar as estruturas de armazenamento propostas neste trabalho, foi desenvolvida a ferramenta PDM (*Palm Database Manager*). Esta ferramenta foi construída sob a arquitetura PDM que é composta pelos seguintes componentes: DBConnLib, DBLib, ConnLib e CompLib. Tais bibliotecas constituem um *framework* que permite trabalhar com a estrutura de armazenamento proposta, compactação de dados e acesso remoto a banco de dados. A biblioteca DBLib possui um conjunto de funções específicas para gerenciar os dados através de uma visão relacional. Para manipular os dados em formato de compressão, foi citada a biblioteca CompLib. Para viabilizar o acesso a banco de dados de forma remota foram definidas as bibliotecas DBConnLib e ConnLib.

Foram realizados experimentos para avaliar o desempenho dos bancos de dados comprimidos definidos através da estrutura proposta, levando-se em consideração a economia de espaço e os tempos de processamento alcançados. Os resultados obtidos demonstram que a abordagem proposta oferece melhorias, pois ao definir o armazenamento de dados baseado em colunas foi possível disponibilizar uma estrutura mais apropriada para a aplicação das técnicas de compressão de dados. Isto possibilitou a otimização na utilização de espaço de memória e a obtenção de excelentes taxas de compressão de dados que variam de 36% a 52%. Esta variação depende bastante das predominâncias dos tipos de dados existentes nas tabelas.

Dessa forma, é possível concluir que a arquitetura proposta neste trabalho apresenta benefícios como: (1) definição de estruturas que manipulam os dados através de uma visão relacional; (2) facilidade para a criação e manutenção dos esquemas de dados; (3) melhora a performance de operações de busca com filtros verticais (projeções) e consultas com operações de agregação (SUM, MAX ou MIN); (4) definição de mecanismos para garantir a consistência dos dados através da manutenção de restrições de integridade; (5) facilidade na integração entre os dados do dispositivo móvel e do servidor de banco de dados central.

Por fim, a estratégia de armazenamento de dados proposta neste trabalho serviu como base para a definição de técnicas de compressão de dados que atingiram excelentes taxas de compressão quando comparados a trabalhos semelhantes, permitindo otimizar o espaço utilizado em memória e diminuir custo de armazenamento de dados em dispositivos móveis do tipo PDA.

6.2 Trabalhos Futuros

Dentre os trabalhos futuros que podem dar continuidade à proposta aqui apresentada, são apresentados os seguintes tópicos:

- Implementação da estrutura de associação de *chunk* de forma que o armazenamento de dados referentes a uma coluna (atributo) da tabela não esteja limitado ao tamanho máximo de um *chunk*. Dessa forma, quando um *chunk* atingir seu tamanho máximo, um novo *chunk* será criado e associado ao *chunk* que armazena os dados do mesmo atributo. Assim, para armazenar os dados de uma coluna pode existe um conjunto de *chunks* que estão associados logicamente;

- Investigação de novas técnicas para melhorar a performance de consultas sem filtros verticais;

- Migração da biblioteca DBLib para uma plataforma que permita a independência do sistema operacional de forma que as funcionalidades definidas possam ser utilizadas em qualquer ambiente operacional voltado para dispositivos do tipo PDA.

6.3 Publicações

BRAYNER, Ângelo; Pitombeira, Dorotéa Karine. D. e Brito, Ricardo Wagner. C. **Uma Arquitetura Eficiente para Armazenamento, Compressão e Acesso a Dados em Dispositivos Móveis com Recursos Computacionais Limitados.** 20° Simpósio Brasileiro de Banco de Dados, p. 250-264, Uberlândia-MG, 2005.

PITOMBEIRA, Dorotéa K. D.; Brito, Ricardo W. C.; Silva, Wendel. B.; Amorim, Danielle C. C.; Almeida Filho, Maguns P.; Mendes, Marília S.; Oliveira, Wandré A., Carvalho, Thiago L.; Brayner, Ângelo; Mendonça, Nabor C. **PDM: *Palm Database Manager*.** 19° Simpósio Brasileiro de Banco de Dados – I Sessão de Demos, Brasília, 2004.

Referências Bibliográficas

[1] ADDMARK. Disponível em http://www.addamark.com/products/sls.htm. Último acesso em 14/04/2006.

[2] BARBARÁ, D. **Mobile Computing and Database – A Survey**. IEEE Transactions on Knowledge and Data Engineering, vol. 11, no. 1, pp. 108-117, January, 1999.

[3] BLUETOOTH, Inc. Disponível em http://www.bluetooth.com/Bluetooth/Learn/Technology/Specifications. Último acesso em 02/12/2005.

[4] BRAYNER, Ângelo; Pitombeira, Dorotéa K. D. e Brito, Ricardo W. C. **Uma Arquitetura Eficiente para Armazenamento, Compressão e Acesso a Dados em Dispositivos Móveis com Recursos Computacionais Limitados.** 20° Simpósio Brasileiro de Banco de Dados, p. 250-264, Uberlândia-MG, 2005.

[5] BONCZ, Peter et. al.. **MonetDB/X100: Hyper-pipelining Query Execution**. CIDR, 2004.

[6] CHEN, Z.; Gehrke, J e Korn, F. **Query Optimization in Compressed Database Systems**. ACM SIGMOD 2001, 2001.

[7] COPELAND, George et. al. **Data Placement in Bubba**. SIGMOD, 1988.

[8] CÔRTEZ, Sérgio. **Um Modelo de Transações para a Integração de SGBD a um Ambiente de Computação Móvel**. Tese de Doutorado. Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2004.

[9] DAVIS, Gordon B. **Anytime/Anyplace Computing and the Future of Knowledge Work**. Communications of the ACM. Dezembro 2002, p. 67-73.

[10] DUNHAM, Margaret H.; HELAL, Abdelsalam (Sumi). **Mobile Computing and Databases: Anything new?** ACM SIGMORD Record, v. 24, n. 4, Dezembro, 1995.

[11] HUFFMAN, D. **A Method for the Construction of Minimum-redundanc Codes**. In Proc. IRE, 40(9), p. 1098–1101, 1952.

[12] IBM Corporation. **DB2 Everyplace Database.** Disponível em http://www-306.ibm.com/software/data/db2/everyplace/everyplacedb.html. Último acesso em 10/02/2006.

[13] IBM Corporation. **InfoCenter do DB2 Everyplace**. Disponível em http://publib. boulder.ibm.com/infocenter/weahelp/index.jsp?topic=/com.ibm.websphere.db2e.doc/welcome.html. Último acesso em 14/04/2006.

[14] IMIELINSKI, Tomasz; Badrinath, B. R. **Data Management for Mobile Computing**. Communications of the ACM, 37(10):19-28, Outubro, 1994.

[15] KAYAN, Ersan; Ulusoy, Özgur. **Real-Time Transaction Management in Mobile Computing Systems.** 6th International Conference On Database Systems For Advanced Applications, pp. 127-134, 1999.

[16] KX SYSTEMS, Inc. **KDB**. Disponível em http://www.kx.com/products/database.php. Último acesso em 14/04/2006.

[17] LOUREIRO, Antônio A.F.; Sadok, Djamel F.H; Mateus, Geraldo R.; Nogueira, J.M.S.; KELNER, Judith. **Comunicação Sem Fio e Computação Móvel: Tecnologias, Desafios e Oportunidades.** Simpósio Brasileiro de Computação (SBC 2003). Campinas, São Paulo. Agosto. 2003.

[18] MATEUS, Geraldo R.; Loureiro, Antônio A.F. **Introdução à Computação Móvel.** 2004. Disponível em http://homepages.dcc.ufmg.br/~loureiro/cm/docs/cm_livro_2e.pdf. Último acesso em 22/04/2006.

[19] MICROSOFT Corporation. **Introducing Windows Mobile 5.0.** Disponível em http://msdn.microsoft.com/mobility/windowsmobile/howto/windowsmobile5/. Último acesso em 10/02/2006.

[20] MICROSOFT Corporation. **SQL Server 2005 Mobile Edition Features.** Disponível em http://www.microsoft.com/sql/editions/sqlmobile/sqlmobile.mspx. Último acesso em 10/02/2006.

[21] MICROSOFT Corporation. **SQL Server 2005 Mobile Edition Books Online**. 2005. Disponível em http://www.microsoft.com/downloads/. Último acesso em 10/02/2006.

[22] ORACLE Corporation. **Oracle Database Lite10g (10.2.0) Documentation.** 2005. Disponível em http://www.oracle.com/technology/products/lite/tech_papers.html. Último acesso em 25/03/2006.

[23] ORACLE Corporation. **Oracle Database Lite10g: Technical White Paper.** 2005. Disponível em http://www.oracle.com/technology/products/lite/tech_papers.html. Último acesso em 25/03/2006.

[24] ORACLE Corporation. **Oracle Database Lite10g: What's the Difference with other Oracle Database Editions?** 2004. Disponível em http://www.oracle.com/ technology/products/lite/tech_papers.html. Último acesso em 25/03/2006.

[25] ORACLE Corporation. **Oracle Database Lite InDepth.** 2003. Disponível em http://www.oracle.com/technology/products/lite/tech_papers.html. Último acesso em 25/03/2006.

[26] ORACLE Corporation. **Oracle9i Lite: A Technical White Paper.** Disponível em http://www.oracle.com/technology/products/lite/ tech_papers.html. Último acesso em 25/03/2006.

[27] PALMSOURCE, Inc. **Exploring Palm OS: Memory, Databases, and Files**. 2004. Disponível em http://www.palmos.com/dev/support/docs/protein_books.html. Último acesso em 13/04/2006.

[28] PALMSOURCE, Inc. **Exploring Palm OS: Palm OS File Formats.** 2004. Disponível em http://www.palmos.com/dev/support/docs/protein_books/File%20Formats.pdf. Último acesso em 13/04/2006.

[29] PALMSOURCE, Inc. **Palm OS Programmer's API Reference.** 2004. Disponível em http://www.palmos.com/dev/support/docs/palmos. Último acesso em 13/04/2006.

[30] PALMSOURCE, Inc. **Palm OS Programmer's Companion - Volume I**. 2004. Disponível em http://www.palmos.com/dev/support/docs/palmos. Último acesso em 13/04/2006.

[31] PALMSOURCE, Inc. **Palm OS Resource File Formats.** 2004. Disponível em http://www.palmos.com/dev/support/docs/dev_suite/Resource%20File%20Format.pdf. Último acesso em 13/04/2006.

[32] PITOMBEIRA, Dorotéa K. D.; Brito, Ricardo W. C.; Silva, Wendel. B.; Amorim, Danielle C. C.; Almeida Filho, Maguns P.; Mendes, Marília S.; Oliveira, Wandré A., Carvalho, Thiago L.; Brayner, Ângelo; Mendonça, Nabor C. **PDM: Palm Database Manager.** 19° Simpósio Brasileiro de Banco de Dados – I Sessão de Demos, Brasília, 2004.

[33] PITOURA, Evaggelia; Samaras, G. **Data Management for Mobile Computing**. Kluwer Academic Publishers, 1998.

[34] PITOURA, Evaggelia; Samaras, G. **Mobile Computing, In the Encyclopedia of Distributed Computing**. Disponível em: http://www.cs.uoi.gr/~pitoura/distribution/encyclopedia.doc acesso em: 05/12/2005.

[35] SILBERSCHATZ, A.; Korth, H. F. ; Sudarshan, S.. **Sistema de Banco de Dados.** Terceira Edição. São Paulo, McGraw-Hill, 1999. EXCLUIR

[36] STAYANARAYANAN, M.. **Fundamental Challenges in Mobile Computing.** Relatório Técnico. CMU-CS-96-111, Philadelphia, 1999, p. 1-7.

[37] STONEBRAKER, M.; Abadi, D. J.; Batkin, A.; Chen X.; Cherniack, M.; Ferreira, M.; Lau, E.; Lin, A.; Madden, S.; O'Neil, E. J.; O'Neil, P. E.; Rasin, A.; Tran, N. e Zdonik, S. B.. **C-Store: A column-oriented DBMS**. VLDB, 2005.

[38] SYBASE, Inc. **Adaptive Server Anywhere Database Administration Guide.** 2004. Disponível em http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.daptive_server_anywhere_9.0.2/title.htm. Último acesso em 16/04/2006.

[39] SYBASE, Inc. **Introducing SQL Anywhere Studio.** 2004. Disponível em http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.adaptive_server_anywhere_9.0.2/title.htm. Último acesso em 16/04/2006.

[40] SYBASE, Inc. **Sybase IQ**. 2004. Disponível em http://www.sybase.com/products/databaseservers/sybaseiq. Último acesso em 16/04/2006.

[41] SYBASE, Inc. **MobiLink Tutorial.** Disponível em http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.adaptive_server_anywhere_9.0.2/title.htm. Último acesso em 13/04/2006.

[42] SYBASE, Inc. **Synchronization Technologies for Mobile and Embedded Computing.** Disponível em http://www.ianywhere.com/downloads/whitepapers/mobilink_sql.pdf. Último acesso em 13/04/2006.

[43] SYBASE, Inc. **UltraLite Database User's Guide.** 2004. Disponível em http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.adaptive_server_anywhere_9.0.2/title.htm. Último acesso em 16/04/2006.

[44] SYBASE, Inc. **What's New in SQL Anywhere Studio.** 2004. Disponível em http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.adaptive_server_anywhere_9.0.2/title.htm. Último acesso em 16/04/2006.

[45] WESTMANN, T.; Kossmann, D.; Helmer, S. e Moerkotte, G. **The Implementation and Performance of Compressed Databases**, SIGMOD Record 29(3), 2000.

[46] WITTEN, I. H.; Neal, R. e Cleary, J. **Arithmetic Coding for Data Compression.** Communications of the ACM, 30(6), p. 520–540, 1987.

[47] ÖZSU, M. T. e Valduriez, P. (1999). **Principles of Distributed Database Systems.** Prentice Hall.

[48] ZIV, J. e Lempel, A.. **A Universal Algorithm for Sequential Data Compression.** In IEEE Transactions on Information Theory, Vol. 31, No. 3, p. 337-343, 1977.

Apêndice A

API *Database Library* (DBLib)

Este capítulo apresenta a definição das principais funções da Database Library.

Principais Funções da *Database Library* (DBLib)

Nesta seção são apresentadas as funções definidas pela DBLib com o objetivo de disponibilizar condições para criar e manter a estrutura de armazenamento proposta neste trabalho, visando disponibilizar uma visão relacional sobre dados armazenados em arquivos. A Tabela 10 apresenta todas as funções externas da biblioteca DBLib que podem ser utilizadas por aplicações clientes.

Função	**Finalidade**
DBLibAddColumn	Adiciona uma coluna a uma tabela.
DBLibAlterRecord	Altera os valores de um registro da tabela de acordo com uma posição específica e com o índice da coluna a ser alterada.
DBLibCheckColumnForeignKey	Verifica se a coluna é definida como uma *foreign key*. Caso seja, realiza a restrição de integridade referencial.
DBLibCheckColumnPrimaryKey	Verifica se a coluna é uma *primary key* garantindo a restrição de chave primária.
DBLibCheckRelationship	Verifica se uma determinada coluna faz referência a uma chave primária de outra tabela, constituindo uma integridade referencial.
DBLibCheckRelationshipReferences	Verifica se uma coluna está sendo referenciada a partir de uma integridade referencial.

Função	**Finalidade**
DBLibCreateDatabase	Cria um novo banco de dados em um cartão de memória específico, ou seja, é criado um arquivo PDB que armazenará as informações referentes às tabelas daquele banco de dados.
DBLibCreateRelationship	Cria uma integridade referencial entre duas tabelas através de suas colunas.
DBLibCreateTable	Cria uma nova tabela em um banco de dados específico. Esta função chama mais três funções internas: InsertInfTable, InsertInfColumn e CreateColumn e através destas permite armazenar todos os atributos da tabela e de suas colunas.
DBLibCreateUsers	Cria um usuário em um determinado banco.
DBLibCreateView	Cria uma visão sobre uma tabela existente.
DBLibDeleteRecord	Exclui o registro de uma tabela de um banco de dados específico de acordo com a posição do registro na tabela.
DBLibDropColumn	Exclui uma coluna de uma tabela de um banco de dados específico.
DBLibDropDatabase	Exclui um banco dados existente no DBManager.
DBLibDropRelationship	Exclui uma integridade referencial específico de um banco de dados.
DBLibDropTable	Exclui uma tabela de um banco de dados específico.
DBLibDropUsers	Exclui um determinado usuário de um banco de dados específico.
DBLibDropView	Exclui uma determinada visão de um banco de dados específico.

Função	**Finalidade**
DBLibFindRecords	Busca um registro específico de acordo com uma chave de busca em uma tabela de um banco de dados considerando a coluna correspondente ao valor de busca. O retorno desta pesquisa é a posição do valor de busca na tabela. De acordo com esta posição é possível chamar a função interna DBGetRecord e recuperar todos os dados de um determinado registro (coluna). Esta função também é utilizada por todas as funções que necessitam que seja passado como parâmetro a posição do registro na tabela, como por exemplo: DBLibDeleteRecord e DBLibAlterRecord.
DBLibFindRelationship	Busca uma quantidade de registros em uma tabela de um banco de dados específico, em um determinado intervalo.
DBLibGetColumnName	Recupera o nome de uma determinada coluna.
DBLibGetColumnsName	Recupera o nome de todas as colunas de uma tabela.
DBLibGetDatabaseName	Recupera o nome do banco de dados existente no DBManager de acordo com o índice especificado.
DBLibGetDatabasesNames	Recupera os nomes de todos os bancos de dados existentes.
DBLibGetInfColumn	Recupera as informações de uma coluna de uma tabela.
DBLibGetInfColumns	Recupera as informações de todas as colunas de uma tabela.
DBLibGetInfoDatabase	Recupera as informações do banco de dados como a quantidade de tabelas, visões e relacionamentos existentes no banco.
DBLibGetInfoTable	Recupera as informações de uma tabela.
DBLibGetInfoUser	Recupera as informações de um usuário.
DBLibGetInfoView	Recupera as informações de uma visão.

Função	**Finalidade**
DBLibGetNameUser	Recupera o nome de um usuário.
DBLibGetNameUsers	Recupera os nomes de todos os usuários.
DBLibGetNameView	Recupera o nome de uma visão.
DBLibGetNumColumn	Recupera o número de colunas de uma tabela.
DBLibGetNumDatabase	Recupera o número de banco dados existente no DBManager.
DBLibGetNumRecord	Recupera o número de registros de uma tabela.
DBLibGetNumRelationship	Recupera o número de relacionamentos de um banco de dados.
DBLibGetNumTable	Retorna o número de tabelas existentes em um banco de dados específico.
DBLibGetNumUsers	Recupera o número de usuários.
DBLibGetNumView	Recupera o número de visões.
DBLibGetRecord	Recupera os dados de um registro lógico específico de uma tabela, de acordo com a posição do registro na tabela (índice), passando por referência a estrutura que irá armazenar os dados recuperados.
DBLibGetRecords	Recupera os dados dos registros existentes em um determinado intervalo.
DBLibGetRecordView	Recupera o registro lógico de uma determinada visão.
DBLibGetRelationship	Recupera informações de uma integridade referencial entre duas tabelas.
DBLibGetRelationships	Recupera informações de várias integridades referenciais.
DBLibGetTableName	Recupera o nome de uma tabela de um banco de dados específico.
DBLibGetTablesName	Recupera os nomes de todas as tabelas de um determinado banco de dados.
DBLibInsertRecord	Criar um novo registro em uma tabela de um banco de dados específico.

Função	Finalidade
DBLibValidRelationship	Verifica se já existe relacionamento com o mesmo nome passado como parâmetro.
DBLibValidRelationshipEqualFK	Verifica se já existe integridade referencial com os mesmos atributos.

Tabela 10 - Lista com as principais funções externas da DBLib

Na Tabela 11 são listadas as funções internas da BDLib que são utilizadas pelas funções externas, não estando, assim, disponíveis para serem utilizadas por aplicações de usuários.

Função	Finalidade
AdjustString	Função que ajusta uma *string* para o tamanho que se deseja.
AlterInfColumnValues	Função que permite alterar as informações de uma determinada coluna da tabela.
AlterRecord	Função que permite alterar os dados de uma tupla.
CheckColumnForeignKey	Função que verifica se uma coluna é chave estrangeira.
CheckColumnPrimaryKey	Função que verifica se a coluna é chave primária.
CheckFK	Função que recupera o índice correspondente à coluna de chave estrangeira (*FK*) no registro de dados.
CheckFKValues	Função que verifica se o valor que a ser inserido na chave estrangeira já existe na tabela estrangeira.
CheckForeignKey	Função que verifica se existe coluna definida como chave estrangeira.
CheckNameUsers	Função que verifica as informações de um usuário.
CheckNull	Função usada pela DBLibInsertRecord que utiliza a CheckNullCol para verificar se as colunas que possuem seus dados nulos (vazios) podem ter seus valores realmente nulos.

Função	**Finalidade**
CheckNullCol	Função usada pela DBLibInsertRecord e pelo CheckNull. Essa função é utilizada quando um valor inserido for nulo e verifica se a coluna correspondente pode ter valores nulos.
CheckPK	Função usada pela DBLibInsertRecord que verifica se a coluna passada como parâmetro é chave primária.
CheckPKValues	Função usada pela DBLibInsertRecord que verifica se o valor a ser inserido na coluna definida como chave primária já existe na tabela.
CheckPKValuesAlter	Função que verifica os valores da coluna de chave-primária para saber se é possível fazer alterações naquele registro.
CheckPKValuesAlterEqual	Função que verifica se os valores antigos e alterados (novos) da coluna com chave-primária são iguais.
CheckPrimaryKey	Função que verifica se os valores da chave primária podem ser inseridos.
CheckPrimaryKeyAlter	Função que verifica se o valor da chave primária pode ser alterado sem violar a restrição de chave primária.
CheckPrimaryKeyUsedFK	Função que verifica se o valor antigo da chave primária (*PK*) do registro que está sendo alterado permanece igual. (ValorPKAntigo = ValorPKAlterado). A coluna de chave primária deve fazer parte de um relacionamento, tendo assim seus valores referenciados por outra tabela. Caso os valores sejam iguais (ValorPKAntigo = ValorPKAlterado), é permitida a alteração do registro. Caso contrário, é necessário verificar se o valor antigo está sendo referenciado pela outra tabela.
CheckRelationship	Função que verifica se existe relacionamento.
CheckRelationshipFKValues	Função que verifica se aquela chave primária está sendo utilizada por alguma tabela que a referencia.

Função	**Finalidade**
CheckRelationshipPKAllValues	Função que verifica se todos os valores da tabela referenciada existem na tabela da chave primária. Essa função será chamada na tentativa de criação de um relacionamento. Verifica se os valores que já existem, estão presentes na tabela de referência para não deixar o banco inconsistente.
CheckRelationshipReferences	Função que verifica se existe algum relacionamento referenciando uma determinada coluna.
CheckUseForeignKey	Função que verifica se a tabela passada como parâmetro é referenciada por uma relação de integridade.
ConvertTuple	Função que recupera as informações da tupla e converte para seu respectivo tipo. O processo de conversão verifica se foi passado um valor inválido e, caso afirmativo, este valor inválido é convertido para um válido.
CreateColumn	Função utilizada pela DBLibCreateTable que armazena o espaço para a nova coluna da tabela que está sendo criada.
CreateDatabase	Função que cria um novo banco de dados.
CreateRelationship	Função que cria um novo relacionamento entre duas tabelas existentes em um banco de dados.
CreateTable	Função que cria uma nova tabela em um banco de dados.
CreateUsers	Função que cria um novo usuário em um banco de dados.
DeleteInfColumnValues	Função que exclui as informações da coluna a ser excluída.
DeleteRecord	Função que exclui uma determinada tupla de uma tabela do banco de dados.
DeleteValueColumn	Função que exclui os dados de uma coluna da tabela.

Função	**Finalidade**
DmGetType	Função que converte o tipo e retorna o campo que está armazenado em forma de *string*.
DmWriteType	Função que converte de *string* para o tipo passado por parâmetro.
DropColumn	Função que exclui uma determinada coluna de uma tabela.
DropDatabase	Função que exclui um determinado banco de dados.
DropRelationship	Função que exclui um determinado relacionamento.
DropTable	Função que exclui uma determinada tabela.
DropUsers	Função que exclui um determinado usuário.
FindIndex	Função que retorna o índice de uma determinada coluna de uma tabela.
FindRecords	Função que recupera os dados uma determinada tupla.
FindRelationship	Função que retorna o índice do relacionamento passado como parâmetro.
FindTable	Função que retorna o índice da tabela passada como parâmetro.
FindValueColumn	Função usada pela DBLibFindRecord que pesquisa um valor em uma coluna da tabela e retorna a posição deste valor.
FindValueColumns	Função usada pela DBLibFindRecord que pesquisa valores em uma coluna da tabela e retorna as posições destes valores.
FloatToStr	Função que transforma um valor do tipo *float* para o tipo *string* passando como parâmetro o resultado onde será atribuído o valor de retorno com o número de casas decimais.
GetColumnName	Função que recupera o nome de uma coluna de uma tabela.
GetColumnsName	Função que recupera o nome de todas as colunas de uma tabela.
GetColumnType	Função que recupera o tipo de uma coluna da tabela.

Função	**Finalidade**
GetDatabaseName	Função que recupera o nome do banco de dados.
GetInfColumn	Função que retorna a estrutura com as informações de uma determinada coluna.
GetInfColumns	Função que retorna a estrutura com as informações de todas as colunas de uma tabela.
GetInfoDatabase	Função que recupera as informações de um banco de dados.
GetInfoTable	Função que recupera as informações gerais de uma tabela.
GetInfoUser	Função que recupera as informações gerais de um usuário.
GetNameUser	Função que recupera o nome de um usuário.
GetNumColumn	Função que retorna o número de colunas de uma tabela.
GetNumRecords	Função que recupera o número de registros de uma tabela.
GetNumRelationshipByTable	Função que recupera o número de relacionamentos por tabela.
GetRecord	Função que recupera um determinado registro lógico de uma tabela.
GetRecords	Função que recupera os registros de uma tabela de acordo com os parâmetros passados para a função.
GetRelationship	Função que recupera os dados de um relacionamento.
GetRelationshipByIndex	Função que recupera os dados de um relacionamento de acordo com um determinado índice.
GetTableName	Função que recupera o nome da tabela.
GetTablesName	Função que recupera os nomes das tabelas de um banco de dados.
InsertInfColumn	Função utilizada pela DBLibCreateTable que chama N vezes uma outra função: InsertInfColumnValues (onde N corresponde ao número de colunas da tabela que está sendo criada).

Função	Finalidade
InsertInfColumnValues	Função utilizada pela InsertInfColumn que armazena os atributos da nova coluna da tabela que está sendo criada (nome, ID, tipo, tamanho, chave primária e campo nulo).
InsertInfTable	Função utilizada pela DBLibCreateTable que é responsável por criar o registro que irá armazenar as informações da tabela como o nome do banco de dados, ID do banco de dados, nome da tabela, ID da tabela, número de colunas, número de linhas (tuplas) e o último ID da coluna.
InsertRecord	Função que insere o registro de dados em uma tabela.
InsertValueColumn	Função chamada pela DBLibInsertRecord que é utilizada para armazenar um valor referente a uma coluna na tabela.
Power	Função que calcula a potência de um número de acordo com seus parâmetros.
ReturnColPK	Função usada pela DBLibInsertRecord que retorna por parâmetro o nome da coluna que é chave primária.
SetNumRecords	Função usada pela DBLibInsertRecord e DBLibDeleteRecord que atualiza o número de registros da tabela.
TrimRight	Função que retira os espaços à direita de uma variável do tipo *string*.
ValidRelationship	Função que valida uma restrição de integridade.
ValidRelationshipEqualFk	Função que verifica se o relacionamento a ser criado já existe.

Tabela 11 - Principais funções internas da BDLib

As funções internas foram definidas para permitir modularizar as funções externas possibilitando disponibilizar um maior grau de usabilidade e manutenibilidade das mesmas.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo