RODRIGO KATSUMOTO SAKAI

**EXTENSÃO DE UM SGBD PARA INCLUIR O
GERENCIAMENTO DA INFORMAÇÃO TEMPORAL**

Dissertação apresentada à Escola
Politécnica da Universidade de São
Paulo para obtenção do Título de
Mestre em Engenharia Elétrica.

São Paulo
2007

RODRIGO KATSUMOTO SAKAI

**EXTENSÃO DE UM SGBD PARA INCLUIR O
GERENCIAMENTO DA INFORMAÇÃO TEMPORAL**

Dissertação apresentada à Escola
Politécnica da Universidade de São
Paulo para obtenção do Título de
Mestre em Engenharia Elétrica.

Área de concentração:
Sistemas Digitais

Orientadora:
Profa. Dra. Solange Nice Alves de
Souza

FICHA CATALOGRÁFICA

SAKAI, RODRIGO KATSUMOTO

EXTENSÃO DE UM SGBD PARA INCLUIR O GERENCIAMENTO DA
INFORMAÇÃO TEMPORAL. São Paulo, 2007. 74 p.

Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo.
Departamento de Engenharia de Computação e Sistemas Digitais.

DEDICATÓRIA

AGRADECIMENTOS

Primeiramente a Deus por estar presente em todos os instantes da minha vida, iluminando e guiando meus passos. E que mesmo nos meus momentos de pouca fé foi fonte inesgotável de esperança e apoio.

À minha esposa Simone, pela compreensão, companheirismo e amor que ajudaram a superar mais essa fase de minha vida.

Aos meus pais Ricardo e Heloísa, por todo amor, apoio e incentivo que dedicaram a mim durante toda minha vida.

À minha orientadora, Solange Souza, por acreditar em meu esforço e me transmitir sua experiência e seus conhecimentos através de seus comentários e discussões, que além de ajudar a aprimorar este trabalho, ainda despertaram a inquietação perante as pesquisas.

Ao Laboratório de Tecnologia de Software (LTS) da Escola Politécnica da USP e seus professores, por terem me acolhido oferecendo as condições e recursos necessários ao desenvolvimento dessa pesquisa.

Aos meus amigos, sinceros e leais que sempre estiveram ao meu lado com uma palavra de apoio ou com momentos de descontração.

RESUMO

O fator temporal é uma variável natural da maioria dos sistemas de informação, pois no mundo real os eventos ocorrem de maneira dinâmica, modificando continuamente os valores dos seus objetos no decorrer do tempo. Muitos desses sistemas precisam registrar essa modificação e atribuir os instantes de tempo em que cada informação foi válida no sistema. Este trabalho reúne as características relacionadas aos Bancos de Dados Temporais e Bancos de Dados Objeto-Relacionais. O objetivo primordial é propor uma forma de implementar alguns aspectos temporais, desenvolvendo um módulo que faça parte das características e funcionalidades internas de um SGBD.

O módulo temporal contempla principalmente a parte de restrições de integridade temporal que é utilizada para manter a consistência da informação temporal armazenada. Para isso, é proposto um novo tipo de dado que melhor representa as marcas temporais dos objetos. Uma parte importante para a implementação desse projeto é a utilização de um SGBD objeto-relacional que possui algumas características orientadas a objetos que permitem a extensão de seus recursos, tornando-o capaz de gerenciar alguns aspectos temporais. O módulo temporal desenvolvido torna esses aspectos temporais transparentes para o usuário. Por consequência, esses usuários são capazes de utilizar os recursos temporais com maior naturalidade.

Palavras-chave: Banco de Dados Temporal, Módulo Temporal, Restrições de Integridade Temporal, Tipo de Dado Temporal.

ABSTRACT

The temporal factor is a natural variable of the majority of the information systems, therefore in the real world the events occur in dynamic way, modifying continuously the values of its objects in elapsing of the time. Many of these systems need to register this modification and to attribute the instants of time where each information was valid in the system. This work congregates the characteristics related to the Temporal Databases and Object-Relational Databases. The primordial objective is to consider a form to implement some temporal aspects, developing a module that is part of the characteristics and internal functionalities of a DBMS.

The temporal module mainly contemplates the part of restrictions of temporal integrity that is used to keep the consistency of the stored temporal information. For this, a new data type is proposed that better represent the objects timestamps. An important part for the implementation of this project is the use of a object-relational DBMS that has some object-oriented characteristics that allow the extension of its resources, becoming capable to manage some temporal aspects. The developed temporal module becomes these transparent temporal aspects for the user. For consequence, these users are capable to use the temporal resources more naturally.

Keywords: Temporal Databases, Temporal Module, Temporal Integrity Constraints, Temporal Datatype.

LISTA DE FIGURAS

1	Dimensão temporal no modelo relacional	9
2	Ordem temporal ramificada	12
3	Primitivas temporais	15
4	Exemplo da operação de consulta temporal	21
5	Criação de tabela temporal	25
6	Inserção de dados temporais	26
7	Restrição temporal	26
8	Seleção temporal	27
9	Classe temporal funcionário	29
10	Estrutura do tipo de dado <i>t_perodo</i>	31
11	Classe temporal funcionário com o tipo de dado <i>t_perodo</i>	31
12	Operação de seleção com operadores relacionais	32
13	Operadores de Allen	33
14	Operação de seleção com o operador <i>overlaps</i>	34
15	Tempo de vida dos objetos	36
16	Objeto funcionario com o atributo temporal salário	38
17	Relacionamento entre o objeto departamento e funcionário	40
18	Arquitetura do SGBD PostgreSQL	44
19	Módulo Temporal	46
20	Criando um novo tipo de dado com SQL	47
21	Criando uma tabela com marca temporal	48
22	Código do <i>t_perodo</i>	49
23	Código da função <i>perodo_in</i>	50
24	Módulo Temporal Compilado	51

25	Criação das funções de entrada, saída e o tipo de dado <i>t_perodo</i>	52
26	Implementação das operações temporais	53
27	Criação das funções temporais no SGBD	54
28	Utilização da função <i>periodo_before</i>	55
29	Criação dos operadores	55
30	Utilização do operador <i>before</i>	56
31	Criação do índice para ser utilizado na estrutura <i>t_perodo</i>	57
32	Oleodutos da Petrobras S/A	59
33	Derivados de petróleo dentro de um oleoduto	59
34	Classes do sistema de dutos	60
35	Tabelas do sistema	60
36	Inclusão de novos registros na tabela <i>oleoduto</i>	62
37	Inclusão de novos registros na tabela <i>agendamento</i>	63
38	Operação de seleção com o operador <i>during</i>	63
39	Violação da restrição de chave primária (unicidade)	64
40	Violação da chave primária (nulo)	64
41	Violação da restrição de integridade existencial temporal (<i>before</i>)	65
42	Violação da restrição de integridade existencial temporal (<i>after</i>)	65
43	Violação da restrição de integridade existencial temporal (<i>overlaps</i>)	66
44	Violação da restrição de domínio	66

LISTA DE TABELAS

1	Atributo temporal (relação funcionário)	18
2	Tupla temporal (relação funcionário)	18
3	Inserção de um novo objeto	20
4	Atualização de um objeto	20
5	Exclusão de um objeto	21
6	Restrição existencial - inconsistência temporal	22
7	Dados inconsistentes	24
8	Dados corrigidos	24
9	Operadores temporais	56

LISTA DE ABREVIATURAS

API Application Programming Interface

BD Banco de Dados

MER Modelo Entidade-Relacionamento

RI Restrição de Integridade

RIT Restrição de Integridade Temporal

SBD Sistema de Banco de Dados

SBDT Sistema de Banco de Dados Temporal

SGBD Sistema Gerenciador de Banco de Dados

SGBDOO Sistema Gerenciador de Banco de Dados Orientado a Objetos

SGBDOR Sistema Gerenciador de Banco de Dados Objeto-Relacional

SGBDT Sistema Gerenciador de Banco de Dados Temporal

SQL Structured Query Language

TV Tempo de validade

TV_i Tempo de validade inicial

TV_f Tempo de validade final

SUMÁRIO

1	Introdução	1
1.1	Motivação e Justificativa	2
1.2	Objetivo	5
1.3	Contribuições do Trabalho	6
1.4	Metodologia	6
1.5	Estrutura do Trabalho	7
2	Banco de Dados Temporal	8
2.1	Conceitos dos Aspectos Temporais	9
2.1.1	Tipos de Banco de Dados Temporais	10
2.1.2	Ordem Temporal	12
2.1.3	Variação Temporal	13
2.1.4	Granularidade	14
2.1.5	Representação Temporal	14
2.2	O Modelo de Dados Temporal	16
2.2.1	Modelo de Dados	16
2.2.2	Linguagem de Manipulação	19
2.2.3	Restrição de integridade	21
2.3	Trabalhos Realizados	22
3	Gerência das Informações Temporais	28
3.1	Introdução	28
3.2	O Tipo de Dado " <i>t_perodo</i> "	29
3.2.1	Operadores de Manipulação para <i>t_perodo</i>	32
3.3	Restrições de Integridade Temporal	34

3.3.1	Chave Primária Temporal	34
3.4	Especificação das Restrições de Integridade Temporal	36
3.4.1	Restrição de Integridade Existencial Temporal	36
3.4.2	Restrição de Integridade Referencial Temporal	38
3.5	Operações de Manipulação de Dados e Violação de Restrições de Integridade	39
3.5.1	Operação de Inserção	39
3.5.2	Operação de Remoção e Operação de Atualização	40
4	Extensão das funcionalidades do SGBD para controle da variação temporal	42
4.1	Introdução	42
4.2	Arquitetura do SGBD Temporal	43
4.2.1	Componentes de um SGBD	43
4.3	O Módulo Temporal	46
4.4	Implementação	47
4.4.1	Estrutura do <i>t_perodo</i>	47
4.4.2	Implementação das Operações Temporais	52
4.4.3	Implementação dos Operadores Temporais	55
4.4.4	Implementação das Restrições de Integridade Temporal	56
5	Avaliação do Módulo Temporal	58
5.1	Introdução	58
5.2	Objetos do Sistema	60
5.3	Manipulação dos Dados Temporais	61
5.4	Violação das Restrições de Integridade Temporal	64
6	Conclusão e Trabalhos Futuros	68
6.1	Sobre o Módulo Temporal	68
6.2	Sobre o SGBDOR	69
6.3	Sobre os Bancos de Dados Temporais	69

6.4	Trabalhos Futuros	70
	Referências Bibliográficas	71

1 INTRODUÇÃO

Atualmente, os Sistemas de Bancos de Dados (SBD) das empresas armazenam os dados pertencentes a elas, formando um repositório de onde esses dados são consultados para diversas finalidades (GORALWALLA; OZSU; TANSEL, 1995), que vão desde uma simples consulta ao cadastro de um funcionário até complexos relatórios gerenciais.

Os valores armazenados nesses SBDs representam o estado da empresa em um particular instante. Ou seja, os dados representam somente o estado atual dos objetos que constituem essa empresa. Por exemplo, o preço atual de um produto que a empresa fornece, os dados pessoais de um funcionário ou os dados das vendas efetuadas.

A mudança de estado de um objeto pertencente a este Banco de Dados (BD), ou seja, a atualização dos valores de seus atributos, implica na perda da informação anterior (GORALWALLA; OZSU; TANSEL, 1995; SANTOS, 2003; OZSOYUGLU; SNODGRASS, 1995). Isso acontece porque estes Sistemas Gerenciadores de Banco de Dados (SGBD) não gerenciam a manutenção dos estados anteriores dos objetos. Quando existe essa necessidade, a aplicação ou o usuário é responsável por esse controle.

O aspecto temporal, porém, está naturalmente associado à maioria dos objetos do mundo real sendo que sua manutenção e utilização são importantes para muitas aplicações. Afinal de contas, essas aplicações possuem alguns objetos cujos valores dos atributos são modificados durante sua vida, fazendo com que o objeto em si apresente diferentes estados ao longo de sua existência. Desta forma, essas modificações precisam ser armazenadas, formando o histórico do objeto que mais tarde pode ser consultado. Por exemplo, o objeto produto tem o valor de seu atributo preço modificado ao longo de sua vida e para algumas empresas é necessário armazenar estes valores anteriores com a finalidade de analisar a diferença dos ganhos obtidos em sua venda.

Muitas vezes também é necessário registrar os valores futuros de alguns objetos, identificando situações que ainda não aconteceram, mas podem vir a acontecer por motivos pré-determinados no presente. Por exemplo, a simulação de condições climáticas considerando os fatores e variáveis atuais ou uma simples reserva de um quarto de hotel durante um determinado período futuro.

Além da capacidade de armazenar dados, um Sistema Gerenciador de Banco de Dados

(SGBD) precisa oferecer recursos que possibilitem manipular estas informações de maneira prática e eficiente. Uma linguagem de consulta denominada *Structured Query Language* (SQL) foi proposta (ASTRAHAN; CHAMBERLIN, 1975; ASTRAHAN et al., 1976) para tal finalidade e ao longo do tempo está em contínua evolução. Cada vez mais recursos são adicionados ou melhorados em sua especificação (ISO/IEC, 1999).

Também é necessário que o SGBD gerencie suas operações garantindo que as informações nunca atinjam estados inconsistentes. Assim, os dados devem ser submetidos a um conjunto de restrições que garantam a integridade e consistência dos mesmos.

Para registrar e gerenciar essa variação da informação ao longo do tempo é necessária a implementação e utilização de uma estrutura capaz de associar uma marca temporal aos dados. Desta forma, foi proposta uma estrutura tridimensional (KLASSEN, 1999; SOUZA, 1998) onde a variável tempo é incluída na estrutura plana bidimensional formada por colunas (atributos) e linhas (registros). Nessa estrutura tridimensional, as marcas temporais são vinculadas aos atributos possibilitando o registro dos momentos em que os valores destes atributos são válidos no mundo real.

Desta forma, a estrutura de dados, a linguagem de manipulação e as restrições de integridade devem ser modificadas ou estendidas de maneira a agregar recursos para o tratamento desses aspectos temporais da informação.

Os Bancos de Dados Temporais (BDT) estão sendo pesquisados desde meados de 1970 (EDELWEISS, 1998) para justamente satisfazer estas necessidades de tratamento dos aspectos temporais e nas últimas duas décadas tem tido fortes contribuições com os objetivos de incorporar: novos modelos de banco de dados, incluindo a extensão da linguagem de consulta para manipulação de dados temporais e incorporação das restrições que garantem a consistência da informação (SOUZA, 1998).

De maneira formal, os pesquisadores (GORALWALLA; OZSU; TANSEL, 1995; OZSOYOGU; SNODGRASS, 1995; SOUZA, 1998) definem Bancos de Dados Temporais como sendo aqueles que armazenam todos os estados de uma aplicação (presente, passado, futuro), registrando e gerenciando sua evolução no tempo.

1.1 Motivação e Justificativa

O interesse no gerenciamento desses aspectos temporais da informação é motivado por diversas aplicações práticas. Uma área de aplicação proeminente que pode ser citada como exemplo é o desenvolvimento de sistemas de suporte a decisão (*Decision Support System*) (NJOVU; IBRAHIM, 2003), onde a dimensão temporal é fundamental.

Hoje em dia, dentro das organizações ou empresas, um dos grandes fatores de su-

cesso é a análise dos dados que compõe as informações dessas empresas. Essa análise é realizada para gerar subsídios concretos para as tomadas de decisões estratégicas. Ao mencionar análise de dados é imprescindível associar o elemento tempo, que delimita os acontecimentos em um eixo cronológico e registra a variação dos dados ao longo deste eixo temporal.

Estes dados são representações das informações do mundo real, onde estas informações variam ao longo do tempo. Nesse contexto, é importante para a tomada de decisão, o conhecimento da temporalidade dos dados (WAND; WANG, 1996; CHENGALUR-SMITH; BALLOU; PAZER, 1999; STUMPF; MCDONNEL, 2003). Ou seja, é necessário conhecer o histórico de alguns objetos, onde, agregado aos dados existem valores temporais servindo como marcas temporais que indicam quando esse valor teve validade no ambiente real (aplicação).

Por exemplo, na área de negócios empresariais (vendas e marketing), a estratégia de venda dos produtos pode ser elaborada com base em dados armazenados sobre as vendas efetuadas nos anos anteriores e as promoções que aconteceram durante estas vendas. Para tanto é necessário armazenar os dados sobre as vendas e o momento em que elas foram válidas no passado. O mesmo deve ser feito para os dados relativos às promoções. Assim, torna-se possível o cruzamento dos períodos de validade entre vendas e promoções.

Além desses cruzamentos, é possível ainda qualificar o dado de acordo com sua idade, ou seja, dados mais antigos podem receber um peso menor na hora de classificar a informação, caso o valor do dado para a produção de uma informação varie de acordo com sua idade cronológica (CHENGALUR-SMITH; BALLOU; PAZER, 1999).

É necessário considerar também que os dados armazenados precisam obedecer as restrições impostas pelo ambiente da aplicação modelada (CORDEIRO et al., 2005) e implementadas no SGBD. Por exemplo, o valor de venda de um produto não pode ser inferior a zero ou outro valor maior que zero.

Para essa e outras situações, as restrições de integridade implementadas nos SGBDs relacionais satisfazem as regras de negócio da aplicação no que se refere a manter e garantir a integridade e consistência dos dados.

Por outro lado, com o surgimento de aplicações que precisam armazenar a variação da informação ao longo do tempo, as restrições existentes e implementadas pelos SGBDs não são suficientes para dar o suporte necessário para que a consistência e integridade dos dados continuem sendo respeitadas.

Com esse ambiente, observa-se uma necessidade crescente de mecanismos e ferramentas para apoiar o gerenciamento não somente da informação presente, mas também da informação passada e em alguns casos até mesmo da informação futura.

Assim, com o objetivo de fornecer este novo conjunto de mecanismos e ferramentas, surgiram propostas de modelos de dados temporais (STEINER, 1998; GREGERSEN; JENSEN, 1999; SOUZA, 1998) para auxiliar na representação da dimensão temporal e ainda fornecer uma estrutura adequada para o armazenamento e gerenciamento dessa nova dimensão.

Como partes importantes deste conjunto de ferramentas, também existem propostas para estender ou modificar uma linguagem de manipulação de dados (SNODGRASS et al., 1996a, 1996b; CHEN; KONG; ZANIOLO, 2003; SNODGRASS et al., 1994) capaz de manipular os aspectos temporais dos modelos propostos.

E também como partes desse conjunto de mecanismos, surgiram propostas de restrições de integridade temporal (GERTZ; LIPECK, 1995; BOHLEN, 1994; TANSEL, 2004a; KUNG, 1985; SOUZA, 1998), que se preocupam em manter a consistência dos dados junto com sua evolução no tempo.

Basicamente, o suporte a este novo conjunto de mecanismos e ferramentas que oferecem suporte ao contexto temporal pode ser implementado de três maneiras diferentes:

1. Integralmente na aplicação;
2. Em uma API entre a aplicação e o SGBD e;
3. No próprio SGBD.

Grande parte dos trabalhos realizados (STEINER, 1999; OLIVEIRA, 2003; PINHEIRO, 2003; EDELWEISS et al., 2000) especificam e implementam mecanismos e ferramentas na própria aplicação ou em uma API, ficando desta maneira, a cargo dos analistas e desenvolvedores garantir um mecanismo para o registro e gerenciamento das informações temporais, que incluem uma estrutura de armazenamento, linguagem de acesso e manipulação de dados e restrições de integridade temporais (SOUZA, 1998). Isso permite que nas aplicações possam surgir falhas de integridade temporais, que somente poderão ser percebidas tardiamente.

Além disso, perde-se o conceito de independência de dados proposto na arquitetura de Banco de Dados ANSI (SILBERSCHATZ; KORTH; SUDARSH, 1999), pois parte dos dados são gerenciados pelo SGBD e a outra parte (temporal) é gerenciada pela aplicação ou API, implicando então que a modificação de uma das partes exerça também significativas alterações na outra.

Porém, quando a responsabilidade de gerenciamento do aspecto temporal é retirada da aplicação e transferida totalmente para o Sistema Gerenciador de Banco de Dados, a manipulação destes aspectos torna-se centralizada minimizando os erros de programação, além de torná-los transparentes para os desenvolvedores, facilitando o desenvolvimento da aplicação.

Desta forma, observa-se a necessidade de propor uma solução de implementação dessas restrições de integridade temporal, de maneira que sejam implementadas no próprio SGBD, que passa a ser o único responsável por manter e gerenciar tais restrições.

1.2 Objetivo

O presente trabalho tem como objetivo principal incluir o gerenciamento dos dados temporais assegurando a sua consistência em todos os períodos de existência (passado, presente, futuro) no próprio SGBD. O trabalho apresenta as restrições de integridade temporal encontradas na bibliografia e propõe os mecanismos necessários para sua implementação.

Para tornar possível a implementação dessas características temporais é proposta uma modificação e extensão do SGBD de maneira que seja possível adicionar em seu conjunto de funcionalidades as estruturas e recursos que permitem a manipulação dos dados ao longo do tempo.

Como base para o presente trabalho é utilizado o modelo bitemporal proposto em (SOUZA, 1998). Esse modelo considera que os conceitos de chave primária e chave estrangeira continuem sendo válidos após a agregação dos aspectos temporais. Assim, os dados devem obedecer a um conjunto de regras temporais além das regras convencionais.

No presente trabalho é considerado apenas o tempo de validade. Tempo de validade é a classe de tempo importante sob a ótica da aplicação, já que está associada ao tempo em que a informação é válida no universo de discurso. Para o gerenciamento do tempo de validade dos dados a estrutura *t_peri*do é proposta.

Tempo de transação é importante para a gerência dos dados sob a ótica do SGBD, seu controle por parte do SGBD é deixado para uma segunda etapa, como continuidade do trabalho.

A implementação dos aspectos temporais nesse trabalho não tem a intensão de englobar todas as classes de aplicações temporais. Ela limita-se a alguns aspectos, por exemplo, os intervalos de tempo são definidos como fechados e disjuntos. Esses e outros termos serão descritos ao longo do trabalho. Assim, somente se beneficiarão as classes de aplicações que utilizem esse tipo de intervalo.

Para mostrar a viabilidade da proposta de integrar ao SGBD o controle completo do tempo de validade dos dados temporais, incluindo os aspectos associados ao controle de integridade, é utilizado o sistema gerenciador de banco de dados objeto-relacional (SGBDOR) PostgreSQL. Os SGBDORs são conhecidos por apresentarem algumas características orientadas à objetos como por exemplo, aninhamento de objetos (tabelas) e herança.

O SGBD PostgreSQL foi escolhido por apresentar características que permitem a extensão de funcionalidades, além de possuir características do paradigma de orientação a objetos. Esse SGBD pode ser utilizado gratuitamente segundo a licença BSD (*Berkeley Software Distribution*) e possui o código-fonte aberto, o que permite a extensão de seus módulos e funcionalidades, oferecendo uma maior flexibilidade de desenvolvimento.

1.3 Contribuições do Trabalho

Este trabalho descreve modificações feitas em um SGBD, de forma que esse faça a gerência das informações temporais, incluindo as regras de consistência temporal.

Isso permitiu que todos os aspectos do controle da informação fiquem centralizadas, tirando o ônus dessa gerência dos usuários.

1.4 Metodologia

Para cumprir os objetivos propostos neste trabalho algumas pesquisas foram realizadas tendo como principal fonte de referência os sites específicos da comunidade científica na Internet (ieee.org, acm.org, citeseer) e os materiais bibliográficos (periódicos, revistas, dissertações, livros) disponibilizados na biblioteca da Escola Politécnica da Universidade de São Paulo.

A leitura dos artigos e trabalhos teve bastante enfoque no início do desenvolvimento da dissertação e depois foi diminuindo na medida em que as idéias eram consolidadas, porém essa leitura continuou até o final da dissertação.

As pesquisas em artigos e trabalhos começaram com um enfoque geral e abrangente na área de Banco de Dados Temporais. Depois, conforme a necessidade observada pelo autor, as pesquisas começaram a se direcionar para assuntos mais específicos, passando pela área de modelos temporais, linguagens de consulta temporais e por último a área de restrições de integridade temporal, onde o autor identificou uma carência maior de soluções.

Com isso, foi definido o escopo do trabalho que de maneira abrangente trata os aspectos temporais em SGBD, mas tem como ponto focal as restrições de integridade temporal em SGBDs. O material consultado ajudou na fase de desenvolvimento dos capítulos iniciais (capítulos 1, 2 e 3) da dissertação.

Posteriormente, foi efetuada uma pesquisa na Internet com o objetivo de encontrar ferramentas (SGBD, aplicativos) que tivessem o suporte aos aspectos temporais. Com as ferramentas encontradas (STEINER, 1999; WEISS, 2005) foram realizados testes que

tiveram a finalidade de analisar o conjunto de funções temporais que cada uma oferecia ao usuário.

Para a fase de implementação foi escolhido um SGBD que permitisse a modificação de seu código-fonte e a implementação de módulos que pudessem ser adaptados diretamente no SGBD. Assim, foi escolhido um SGBD gratuito e com o código-fonte aberto.

A fase de implementação foi composta por leituras na documentação do SGBD escolhido, participação em fóruns de discussão e a própria programação dos aspectos temporais. A leitura da documentação teve o intuito de descobrir como poderiam ser realizadas as implementações necessárias e os fóruns de discussão auxiliaram a resolver as dúvidas pertinentes a utilização e programação do SGBD escolhido.

A parte inicial da implementação consolidou o enfoque do trabalho, determinando como seriam realizados os próximos passos da implementação e desenvolvimento da dissertação. Assim, foi finalizado o capítulo 3 e desenvolvido o capítulo 4 da dissertação.

Para a fase de testes foi elaborado um exemplo básico utilizando os aspectos temporais implementados na fase anterior. Nessa fase, foram avaliadas as características funcionais do projeto com o intuito de comprovar sua utilidade e encontrar pontos de melhoria que foram incorporados no trabalho ou que foram documentados para serem incluídos em trabalhos futuros. Os testes efetuados compuseram o capítulo 5 deste trabalho.

Por fim, todas as considerações levantadas no desenvolvimento desta dissertação foram sendo separadas ao longo do tempo para compor o desfecho do trabalho.

1.5 Estrutura do Trabalho

Este trabalho encontra-se estruturado da seguinte forma. O capítulo 2 apresenta uma revisão bibliográfica abrangendo os principais conceitos sobre Sistemas de Bancos de Dados Temporais (SBDT), também apresenta os aspectos importantes de outros trabalhos, destacando-se iniciativas de implementação de modelos temporais. O Capítulo 3 apresenta as propostas para gerência das informações temporais incluindo estruturas e operações temporais necessárias à gerência. O capítulo 4 apresenta uma proposta de extensão de um SGBD para a implementação dos aspectos temporais. O capítulo 5 procura demonstrar por meio de um exemplo, a validade da implementação proposta. E para finalizar, o capítulo 6 apresenta a conclusão final do trabalho e perspectivas futuras.

2 BANCO DE DADOS TEMPORAL

A maioria das aplicações são naturalmente temporais (JENSEN; SNODGRASS, 1999) registrando diferentes estados para seus objetos ao longo do tempo. Em alguns destes casos é interessante e necessário que este histórico da informação seja preservado. Por exemplo, em aplicações financeiras é necessário manter armazenado a variação dos valores de uma conta corrente ou poupança ao longo do tempo, assim também em aplicações científicas sobre a variação climática durante um período e aplicações para gerenciamento de um hotel onde é registrado até mesmo períodos futuros de reservas de apartamentos.

O termo temporal refere-se ao passado, presente e futuro de alguma coisa. Uma informação temporal ou dado temporal inclui duas partes: (a) o estado de uma entidade e (b) o intervalo de tempo da informação na qual este estado é verdadeiro. Lembrando que o estado de uma entidade é definido pelo conjunto de valores que os atributos da entidade possui em um determinado intervalo de tempo. O banco de dados que mantém os dados passado, presente e futuro armazenados é chamado de Banco de Dados Temporal (BDT) (OZSOYUGLU; SNODGRASS, 1995) e há mais de duas décadas que este assunto sobre o aspecto temporal vem sendo objeto de estudo de vários pesquisadores da área gerando algo em torno de 2.000 artigos escritos listados em (KLINE, 1993), alguns livros (JENSEN; SNODGRASS; TANSEL, 1994; SNODGRASS, 1999; DATE; DARWEN; LORENTZOS, 2002) e um glossário de terminologias (JENSEN et al., 1992).

Os bancos de dados temporais possuem outras denominações aceitas na comunidade, mas com algumas considerações. Eles também são referenciados como *time-oriented databases*, *time-varying databases* e *historical databases*. Realmente as duas primeiras denominações são equivalentes a banco de dados temporais, mas a última não é, pois trata somente dos dados históricos, ou seja, dos dados passados, constituindo apenas uma parte do conceito de banco de dados temporais.

O principal fator em banco de dados temporal é a necessidade de adicionar a terceira dimensão na estrutura bidimensional dos bancos de dados instantâneos (Figura 1). Onde esta terceira dimensão representa os valores temporais de cada objeto, ou seja, cada estado de um objeto está representado durante períodos ou intervalos de tempo. Mas este fator leva a uma modificação no conceito de bancos de dados relacionais, sendo necessário

Figura 1: Dimensão temporal no modelo relacional

estender o modelo relacional para que continue valendo os conceitos de: estrutura de dados, restrições de integridade e linguagem de manipulação propostas por (CODD, 1970).

2.1 Conceitos dos Aspectos Temporais

O tempo é uma entidade que possui características próprias e a boa compreensão do seu significado é muito importante para sua manipulação (SOUZA, 1998). Muitos dos objetos de uma empresa, que são representados nos Bancos de Dados, têm aspectos temporais agregados a ele, representando os estados desses objetos. Dentre esses aspectos, são destacados as classes temporais: tempo válido, tempo de transação e tempo definido pelo usuário. Estas denominações e os conceitos seguintes seguem o glossário definido em (JENSEN et al., 1992).

- A classe de **tempo válido** (TV) representa o momento (presente, passado, futuro) em que a informação foi, é ou será verdadeira no mundo real. Pode ser definido por meio de duas marcas temporais que delimitam o intervalo de validade do objeto: o tempo válido inicial (TV_i) e o tempo válido final (TV_f). Por exemplo, é possível dizer que o preço de determinado produto foi de R\$ 30,00 desde 20 de março de 2005

até 31 de dezembro de 2005, ou seja, $TV_i = [20/03/2005]$ e $TV_f = [30/12/2005]$. Todo objeto do mundo real tem um tempo válido, no entanto muitas vezes não é necessário armazenar essa informação temporal, ou porque o tempo é desconhecido ou porque é irrelevante para a aplicação (JENSEN; SNODGRASS, 1999). O tempo válido é manipulado diretamente pelo usuário, sendo dele a responsabilidade de atribuir valores para essa classe de tempo. Indistintamente, o termo tempo de validade pode ser utilizado para se referir ao tempo válido de um objeto.

- A classe de **tempo de transação** (TT) representa o momento em que determinado fato é registrado no banco de dados, ou seja, o momento em que uma operação ou transação foi efetuada no Banco de Dados (BD), também pode ser delimitado por duas marcas de tempo: o tempo de transação inicial (TT_i) e o tempo de transação final (TT_f). Seguindo o mesmo exemplo anterior, o fato do produto custar R\$ 30,00 a partir do dia 20 de março de 2005 foi registrado no dia 20 de março de 2005 e o fato que este valor durou até 30 de dezembro de 2005 foi registrado no dia 1 de Janeiro de 2006, ou seja, $TT_i = [20/03/2005]$ e $TT_f = [01/01/2006]$. O tempo de transação deve ser gerenciado pelo SGBDT e é limitado pelo instante de criação da base de dados e o instante atual.
- **Tempo definido pelo usuário** é considerado como qualquer outro atributo cujo domínio é representado por um tipo de dado de tempo (data, hora). Por exemplo, a data da compra de algum produto é um atributo com tempo definido pelo usuário, em que o SGBD não precisa exercer nenhuma gerência além das convencionais sobre o domínio de data.

Apesar de muitas vezes os tempos de validade e de transação de um objeto se sobrepor, eles são totalmente ortogonais, ou seja, são tratados independentemente um do outro. Assim poderia ter sido registrado hoje (Tempo de Transação) que determinada entidade teve seus valores válidos no mês anterior (Tempo de validade).

Para a manipulação dessas classes temporais é necessário a criação de modelos e estruturas de dados que conferem ao Banco de Dados as características temporais.

2.1.1 Tipos de Banco de Dados Temporais

Um Banco de Dados Temporal pode implementar uma dessas classes temporais, nenhuma ou as duas classes devido a independência temporal entre elas. Desta maneira, as seguintes categorias de BD são identificadas:

- Banco de Dados Instantâneo ou Snapshot

- Banco de Dados de Tempo de Transação ou Rollback
- Banco de Dados de Tempo de Validade ou Histórico
- Banco de Dados Bitemporal

Os **Bancos de Dados Instantâneos** armazenam somente as informações atuais, não possuindo nenhuma característica temporal além dos atributos de tempo definidos pelo usuário. Desta maneira, a cada modificação dos dados ocorre uma transição de estado, onde os dados anteriores são completamente perdidos. Em alguns trabalhos antecedentes (SANTOS, 2000, 2003; PINHEIRO, 2003), os pesquisadores têm chamado de bancos de dados convencionais aqueles que não tratam os aspectos temporais, mas segundo (JENSEN et al., 1992) essa denominação deixa de ter um significado claro na medida que o tempo passa e a tecnologia evolui. Por isso, será utilizado neste trabalho o termo banco de dados *snapshot* ou instantâneo, para se referir aos bancos de dados que armazenam somente informações atuais.

Os **Bancos de Dados de Tempo de Transação** suportam apenas a classe de tempo de transação em sua estrutura. Então, são capazes de armazenar os instantes de tempo em que uma transação foi efetivada e o dado foi armazenado no BD. Assim, torna-se possível desfazer certas operações a fim de retornar o BD a um estado consistente, anterior a uma falha do sistema ou falha humana ou ainda recuperar informações antigas. Por isso que ele também é conhecido como Banco de Dados de *Rollback*. Lembrando que o conceito de transação efetivada refere-se a um conjunto de operações concluídas sem ferir as propriedades de Atomicidade, Consistência, Integridade e Durabilidade (ACID)¹.

Os **Bancos de Dados de Tempo Válido** suportam apenas a classe de tempo válido dos dados, ou seja, nesse tipo de BD é possível associar aos dados o tempo em que a informação correspondente é válida no mundo real. Com o tempo de validade é possível representar o tempo passado, o tempo presente ou o tempo futuro em que a informação será válida. Para esse tipo, a denominação de Banco de Dados Histórico também pode ser utilizada, mas apenas quando não há suporte para dados futuro.

Para ser chamado de **Banco de Dados Temporal** ou bi-temporal é necessário que o SGBD implemente o tratamento das classes temporais: tempo de transação e tempo válido. Desta maneira, constitui-se um Banco de Dados onde é possível armazenar os dados temporais do mundo real e seu tempo de efetivação no BD. Atualmente, nenhum dos SGBDs comerciais tratam dos dois aspectos temporais. Alguns implementam a classe de tempo de transação, em (ORACLE, 2002; WEISS, 2005) é mostrada a funcionalidade *FlashBack* do Oracle onde, a partir da versão 9i, o SGBD tem a capacidade de registrar o tempo de transação de cada operação efetivada, tornando possível desfazer um conjunto

¹Propriedade do SGBD que garante a consistência dos dados após a efetivação de uma transação

de operações baseadas em um determinado momento e até mesmo efetuar consultas em estados anteriores do banco de dados.

2.1.2 Ordem Temporal

A variação dos dados ao longo do tempo é composta por uma sequência de pontos temporais consecutivos e recebe o nome de eixo temporal (EDELWEISS, 1998; SOUZA, 1998). Neste eixo temporal os valores definidos para TT e TV devem estar em ordem cronológica onde esta pode ser:

- **Ordem Linear** - Dado um ponto temporal T , existe apenas um ponto antecessor T_a e um ponto sucessor T_s . Assim, dado o operador *before* equivalente à operação relacional "<" (menor que) sobre os números inteiros, é possível definir, T_a *before* T *before* T_s . Por exemplo, o salário de um funcionário ao longo do tempo pode ser representado por uma sequência linear com um único valor imediatamente anterior ao valor atual e um único valor imediatamente posterior ao valor atual. Esse tipo de ordem é a forma mais usual de ordenação temporal.
- **Ordem Ramificada** - Dado um ponto no tempo T , pode existir mais de um elemento sucessor e/ou mais de um elemento antecessor, ou seja, um objeto pode possuir mais de um valor histórico possível (passado ramificado) e/ou mais de um valor futuro possível (futuro ramificado). Por exemplo, hipóteses evolucionárias do universo caracterizando um passado ramificado, onde para um mesmo período passado existam informações referentes a hipóteses diferentes. Como exemplo de futuro ramificado, pode-se citar as condições possíveis que um sistema pode assumir dependendo das variáveis atuais caracterizando um futuro ramificado. Esta ordem está ilustrada na (Figura 2)

Figura 2: Ordem temporal ramificada

- **Ordem Circular** - Um conjunto de eventos se repete periodicamente em determinada ordem. Por exemplo, eventos que ocorrem de acordo com datas festivas do ano (carnaval, páscoa, natal, etc). Na ordenação circular o tempo não anda de maneira repetitiva, na realidade o que se repete ao longo do tempo são os eventos.

Como dito anteriormente, a ordenação linear é a mais utilizada atualmente nos modelos de dados. A ordenação linear no tempo pode ser melhor definida por meio da teoria dos conjuntos mostrada a seguir.

Dado um conjunto não vazio $\mathbf{T}$ com todos os pontos no tempo. Qualquer ponto nesse conjunto deve satisfazer a seguinte condição (EDELWEISS, 1998):

$$\forall t_a, t_b: t_a, t_b \in T \wedge t_a \neq t_b \longrightarrow (t_a \text{ before } t_b \vee t_b \text{ before } t_a)$$

Além disso, as seguintes propriedades devem ser respeitadas:

- Irreflexibilidade: Um ponto t no tempo (T) não pode ser posterior ou anterior a ele mesmo.

$$\forall t: t \in T \longrightarrow \neg (t \text{ before } t)$$
- Transitividade: Dado três pontos no tempo t_a, t_b, t_c e se t_a é anterior a t_b e se t_b é anterior a t_c , então t_a é anterior a t_c .

$$\forall t_a, t_b, t_c: t_a, t_b, t_c \in T \wedge t_a \text{ before } t_b \wedge t_b \text{ before } t_c \longrightarrow t_a \text{ before } t_c$$
- Assimetrias: Dado dois pontos t_a e t_b e se t_a é anterior a t_b então t_b não pode ser anterior a t_a .

$$\forall t_a, t_b: t_a, t_b \in T \wedge t_a \text{ before } t_b \longrightarrow \neg (t_b \text{ before } t_a)$$

Neste trabalho está sendo considerada apenas a ordenação linear, pois ela é mais adequada a grande parte das aplicações, além de tornar mais simples a proposta de implementação.

2.1.3 Variação Temporal

O tempo é **contínuo** por natureza, onde entre dois instantes consecutivos temporais sempre pode existir um outro instante temporal, o modelo contínuo é isomorfo aos números racionais. Mas, sem perder a semântica temporal é possível representar o tempo de maneira **discreta**, onde dado dois pontos consecutivos no eixo temporal não existe nenhum outro ponto entre eles. O tempo discreto é utilizado neste trabalho de forma a simplificar o modelo de dados e sua implementação. Este conceito de tempo discreto se baseia em

uma linha de tempo (eixo temporal) composta de uma sequência de intervalos temporais consecutivos que não podem ser decompostos (EDELWEISS, 1998). O intervalo entre dois pontos temporais consecutivos é indivisível e denominado *chronon*.

2.1.4 Granularidade

A granularidade é uma propriedade que especifica a unidade de medida (segundo, minuto, hora, dia, ano, etc) do tempo (SOUZA, 1998). Ela consiste na duração de um *chronon*, que é a menor unidade de tempo (valor atômico) que pode ser representada no modelo e não pode ser decomposta.

As classes de tempo válido e de tempo de transação possuem granularidades específicas diferentes para cada um. Em particular o tempo de transação tem uma granularidade única para todo o SGBD, assim no momento da criação de um Banco de Dados deve ser informado qual a unidade de medida a ser utilizada, priorizando uma granularidade de segundos ou até menor, pois em um ambiente multiusuário, várias transações podem estar ocorrendo ao mesmo tempo.

Para a classe de tempo válido, a granularidade pode ser especificada diferentemente para cada objeto criado, isso representa mais fielmente a realidade, pois os objetos no mundo real podem ter seus atributos variando em espaços de tempo distintos. Por exemplo, os preços de um produto variam em intervalos de dias e o salário dos funcionários em meses.

Dado a possibilidade de existir granularidades diferentes dentro do SGBD é necessário implementar funções de conversão para conseguir relacionar objetos com diferentes granularidades.

2.1.5 Representação Temporal

O tempo pode ser representado basicamente por três primitivas temporais:

- **Instante Temporal** - Considerando o tempo discreto, o instante temporal é representado por um *chronon* do eixo temporal suportado pelo modelo. Na variação discreta entre dois pontos consecutivos no tempo não existe nenhum outro ponto no tempo. Considerando ainda a ordenação linear, existe um ponto especial denominado **instante atual** (*now*), que se movimenta constantemente ao longo do tempo e todo ponto anterior a ele é considerado passado e todo ponto posterior a ele é considerado futuro.
- **Intervalo Temporal** - Considerando o tempo discreto, o intervalo temporal é definido como um conjunto finito de *chronons* consecutivos formando assim um subcon-

junto do eixo temporal do modelo. Esse conjunto é delimitado pelo instante inicial e pelo instante final da classe temporal. Esses pontos limites podem pertencer ou não ao intervalo, assim é possível definir: intervalo aberto, quando nenhum dos pontos limites pertencem ao intervalo; intervalo semiaberto, quando um dos dois pontos limites pertencem ao intervalo; e intervalo fechado, quando os dois pontos limites pertencem ao intervalo. Matematicamente é possível representar:

- Intervalo aberto: Os colchetes voltados para fora da sentença e os parênteses indicam onde o intervalo é aberto. Nesse caso, o intervalo é aberto nos dois limites (instante inicial e final). Assim, os dois colchetes estão voltados para fora da sentença.

$]t_1, t_2[$ ou (t_1, t_2)

- Intervalo semiaberto: Os colchetes voltados para dentro da sentença indicam onde o intervalo é fechado, os colchetes voltados para fora e os parênteses continuam indicando onde o intervalo é aberto. Assim, na primeira sentença, o intervalo é aberto no limite (instante) inicial e fechado no limite final.

$]t_1, t_2] \vee [t_1, t_2[$ ou $(t_1, t_2] \vee [t_1, t_2)$

- Intervalo fechado: Como os dois colchetes estão voltados para dentro da sentença, o intervalo é fechado no limite inicial e no limite final.

$[t_1, t_2]$

Onde, t_1 é o primeiro ponto do intervalo, t_2 é o último ponto e $t_1 \leq t_2$.

- **Elemento Temporal** - A união finita de intervalos temporais é representada pelo elemento temporal, ou seja, um elemento temporal é formado por um conjunto finito de intervalos temporais, podendo esses conjuntos serem disjuntos. Na representação de dados, o elemento temporal pode ter uma semântica melhor que as outras duas primitivas anteriores, pois o elemento temporal evita que os dados sejam fragmentados em várias tuplas representadas por intervalos temporais.

<u>Salario</u> 1.800,00 [01] 1.800,00 [10] 2.500,00 [16]	<u>Salario</u> 1.800,00 [01, 05) 1.800,00 [10, 16) 2.500,00 [16, now)	<u>Salario</u> 1.800,00 {[01, 05), [10, 16)} 1.800,00 {[16, now)}
(a)	(b)	(c)

Figura 3: Primitivas temporais

A figura 3 mostra respectivamente essas três diferentes formas de representar o tempo, onde os valores do tempo (marcas temporais) estão expressos em números inteiros para

simplificar o exemplo. Esses números estão dentro de colchetes ou parênteses indicando os tipos de intervalos que cada valor está associado.

A semântica temporal é melhor representada pelo intervalo de tempo (Figura 1b), pois consegue mostrar mais facilmente o período em que cada dado é efetivamente válido na aplicação. Na figura 1(b) é possível verificar que no instante [14] o salário do funcionário era de R\$ 1.800,00.

Utilizando-se o instante temporal (Figura 1a) não é muito claro identificar todos os instantes em que o dado foi válido. Observando a figura 1(a), não é possível dizer que no instante [6] não havia nenhuma informação de salário válida.

O elemento temporal agrupa em um único registro todo o histórico de um dado, além de conseguir representar o instante em que cada dado é válido. Apesar disso, essa primitiva temporal dificulta a manipulação dos dados, pois devem ser realizadas operações de desdobramento (unfold) antes de utilizar a informação. Por exemplo, no primeiro registro da figura 1(c) existem dois intervalos de tempo para o mesmo objeto, assim deve-se realizar uma operação para separar estes dois intervalos antes de efetuar qualquer outra operação de comparação ou relacionamento

2.2 O Modelo de Dados Temporal

O gerenciamento de dados temporais pode se tornar muito difícil quando são utilizados modelos e linguagens de consulta não temporais (JENSEN; SNODGRASS, 1999), como por exemplo o modelo relacional e a linguagem SQL ANSI.

Como mencionado no início deste capítulo, a implementação das características temporais descritas nas seções anteriores torna necessário estender o modelo de dados adicionando uma terceira dimensão em sua estrutura. A inclusão dessa dimensão temporal reflete em modificar ou estender três definições: o modelo de dados, a linguagem de consulta para manipular este modelo e as restrições de integridade.

2.2.1 Modelo de Dados

O modelo de dados é utilizado para organizar as informações de parte da realidade que se deseja representar. Os modelos de dados podem ser considerados em dois contextos (JENSEN; SNODGRASS, 1999). No primeiro contexto, os **modelos conceituais** representam as informações com um elevado grau de abstração preocupando-se com a semântica da aplicação, mantendo-se desta maneira totalmente independente do Sistema Gerenciador de Banco de Dados, o modelo entidade relacionamento proposto por (CHEN, 1976) é o modelo conceitual mais utilizado atualmente. No segundo contexto, os **modelos lógicos**

implementam essa representação por meio de uma definição formal das estruturas lógicas de armazenamento, assim tornam-se dependentes do tipo de Sistema Gerenciador de Banco de Dados, o modelo relacional definido por (CODD, 1970, 1990) é o modelo lógico utilizado e implementado pela maioria dos SGBDs comerciais hoje em dia.

Basicamente, cada modelo de dados é diferenciado pela estrutura lógica de dados que é implementada e pela linguagem utilizada para manipular essas estruturas.

Em relação ao modelo de dados conceitual temporal, alguns esforços se concentram no modelo entidade-relacionamento (MER) onde algumas notações foram propostas e definidas e uma relação das mais significativas pode ser encontrada em (GREGERSEN; JENSEN, 1999). Nesse artigo, os pesquisadores defendem diferentes abordagens destacando-se aquelas que modificam totalmente a notação conceitual e as estruturas de representação dos dados e aquelas abordagens que estendem a notação do modelo entidade-relacionamento para adicionar a capacidade de representar os aspectos temporais da informação.

O modelo conceitual temporal considerado ideal é aquele que possua fácil compreensão, não invalide modelos e aplicações anteriores (sistemas legados) e principalmente que permita ao projetista do Banco de Dados misturar os aspectos temporais e não temporais (JENSEN; SNODGRASS, 1999), pois uma aplicação muitas vezes possui apenas uma parte temporal.

Em relação ao modelo lógico e físico de dados, onde os detalhes de implementação já são especificados, é necessário que seja definida uma estrutura de armazenamento eficiente, onde a variação da informação possa ser armazenada e manipulada. A estrutura bidimensional dos Bancos de Dados relacionais instantâneos não comporta esta variação da informação, sendo necessária sua modificação.

A modificação ou extensão dessa estrutura de dados é o primeiro passo para oferecer suporte para o gerenciamento de dados temporais (JENSEN; SNODGRASS, 1999). Para representar os aspectos temporais, a estrutura de dados deve ser capaz de armazenar a variação da informação ao longo do tempo. Assim, é necessário agregar a cada entidade temporal atributos que armazenem o tempo de transação (TT) e/ou o tempo válido (TV).

Para a modificação dessa estrutura as primitivas temporais apresentadas na seção 2.1.5 podem ser adicionadas ao contexto do SGBD conforme uma das duas principais abordagens. Na primeira abordagem são utilizadas relações normalizadas pela primeira forma normal (1FN) onde as primitivas temporais são adicionadas em cada tupla de uma relação definindo a tupla temporal (*tuple timestamping*). Na segunda abordagem são utilizadas relações que não estão na primeira forma normal possuindo atributos multivalorados onde as primitivas temporais estão associadas a cada atributo que tenha seus valores modificados ao longo do tempo, definindo os atributos temporais (*attribute timestamping*) (TANSEL, 2004b; GORALWALLA; TANSEL; ÖZSU, 1994; GORALWALLA; ÖZSU;

TANSEL, 1995).

A tabela funcionário (tabela 1) apresenta o conceito de atributo temporal para representar o tempo de validade, onde a variação dos valores do atributo salário é registrada. Observa-se na tabela que o funcionário Roberto tem dois salários associados a ele, o primeiro foi válido de janeiro de 1997 até junho de 1999 e o segundo salário é válido desde julho de 1999 até os dias atuais. No mesmo exemplo a funcionária Andrea possui um único registro de salário com o valor de TV_f definido no passado indicando que esta funcionária fez parte da empresa no período [Set 97, Set 99) e provavelmente não tem mais vínculo com a empresa.

Fcodigo	Fnome	Fsalario
001	Roberto	<[Jan 97, Jul 99), 2.600>, <[Jul 99, <i>now</i>), 2.900>
002	Caetano	<[Mar 97, Dez 99), 1.600>, <[Dez 99, <i>now</i>), 2.100>
003	Andrea	<[Set 97, Set 99), 2.600>

Tabela 1: Atributo temporal (relação funcionário)

O mesmo exemplo da tabela 1 pode ser representado utilizando a abordagem de tupla temporal (*tuple timestamping*) de forma que os valores para o tempo de validade inicial (TVi) e tempo de validade final (TVf) sejam representados por dois outros campos no objeto. Desta forma, é necessário que cada registro seja representado por tuplas com atributos mono-valorados. A tabela 2 demonstra o mesmo exemplo utilizando tupla temporal.

Fcodigo	Fnome	Fsalario	TVi	TVf
001	Roberto	2.600	Jan 97	Jul 99
001	Roberto	2.900	Jul 99	<i>now</i>
002	Caetano	1.600	Mar 97	Dez 99
002	Caetano	1.600	Dez 99	<i>now</i>
003	Andrea	2.600	Set 97	Set 99

Tabela 2: Tupla temporal (relação funcionário)

A utilização de atributo temporal tem a vantagem de que todos os dados temporais referentes a um objeto são representados em um único registro tornando mais fácil a visualização de toda informação pertinente a este objeto, além de evitar a redundância repetindo dados que não variam no tempo, como por exemplo o nome do funcionário. No caso de utilizar a abordagem de tupla temporal a informação relacionada a um único objeto, no caso um funcionário, pode estar espalhada por vários registros dificultando a visualização das informações e ainda torna necessária a repetição da informação.

Em contrapartida, a abordagem de tupla temporal torna mais fácil a manipulação

dessas informações, pois para efetuar uma consulta que retorne os funcionários em um determinado período, por exemplo [Jan 99, Jan 00), basta comparar os períodos válidos de cada tupla verificando se estão dentro do período pedido, no caso de utilizar o atributo temporal é necessário antes efetuar uma operação que desagrupe os atributos multivalorados para depois comparar cada período individualmente.

2.2.2 Linguagem de Manipulação

A **linguagem de manipulação** deve ser enriquecida para poder trabalhar com a dimensão temporal facilitando a manipulação da informação. Assim, as instruções de criação, alteração e remoção de objetos (*Data Definition Language - DDL*) e as operações de inserção, atualização, exclusão e consulta de dados (*Data Manipulation Language - DML*) devem ser estendidas para que os aspectos temporais possam ser manipulados. Para isso, pesquisas (SNODGRASS; KUCERA, 1995; SNODGRASS et al., 1996a, 1996b) foram feitas no sentido de criar ou modificar uma linguagem capaz de manipular um modelo temporal, ao todo cerca de 40 linguagens foram propostas (JENSEN; SNODGRASS, 1999) sendo que a linguagem *Temporal Structured Query Language 2* (TSQL2) proposta por (SNODGRASS et al., 1994) é uma tentativa de consolidar as melhores definições de cada uma. A maior parte das linguagens de manipulação de dados temporais propostas, inclusive a TSQL2, se baseiam na *Structured Query Language* (SQL) pois ela é a linguagem de SGBDs mais utilizada atualmente.

Para uma completa linguagem de manipulação de informações temporais, as seguintes características básicas foram estabelecidas (EDELWEISS, 1998):

- Suporte a intervalos de tempo e elementos temporais. Em SQL padrão, existe somente *date*, *time* e *timestamp*.
- Suporte a múltiplas granularidades, por exemplo semanas, semestres e estações do ano. Em SQL padrão, existe somente *year*, *month*, *day*, *hour*, *minute* e *second*.
- Suporte a múltiplos idiomas. Por exemplo, inglês e português.
- Suporte a múltiplos calendários (lunar, acadêmico, fiscal, chinês, etc).
- Suporte a tempo indeterminado. Por exemplo, "Entre 20 e 28 de agosto".
- Suporte a tempo histórico. Consulta a estados passados dos objetos.

Além dessas características básicas, é necessário que as operações sejam modificadas para que o contexto histórico seja preservado. No caso da DDL, deve ser possível especificar e criar objetos com características temporais, por exemplo, criar uma tabela com

tempo de validade. No caso da DML, as operações de inserção (*INSERT*), atualização (*UPDATE*) e remoção (*DELETE*) tomam um aspecto bem diferente, devido ao fato de que em um Banco de Dados Temporal são mantidos os valores atuais e históricos para cada objeto. Assim, de acordo com (SOUZA, 1998) e considerando apenas o tempo de validade dos objetos, é possível especificar resumidamente que:

Na operação de **inserção**, o usuário deve também atribuir valores para os atributos temporais (tempo de validade). Como a operação de inserção adiciona um novo registro ao objeto, não é necessário preocupar-se com valores anteriores para este objeto. É possível não informar o valor do tempo de validade inicial (TV_i) nem o valor do tempo de validade final (TV_f), assumindo assim que $TV_i = [now]$ e $TV_f = [null]$, indicando que os valores desse novo objeto são válidos a partir do momento de inserção no Banco e continuarão sendo válidos até que novos valores sejam definidos (atualização) ou que o objeto deixe de existir (remoção).

Para o caso do usuário digitar os valores do tempo de validade do objeto, deve sempre ser verificado que $TV_i \leq TV_f$, ou seja, que o tempo de validade inicial seja sempre menor ou igual ao tempo de validade final. A tabela 3 mostra o estado de um objeto que foi incluído com seu tempo de validade inicial igual a 20 de janeiro de 2005 ($TV_i = [20/01/2005]$) e seu tempo de validade final indeterminado ($TV_f = [null]$).

Fcodigo	Fnome	Fsalario	TVi	TVf
001	Roberto	2.600	Jan 2005	<i>null</i>

Tabela 3: Inserção de um novo objeto

Na operação de **atualização** ocorre na verdade uma inserção de um novo registro com os valores novos e o tempo de validade final do registro sendo modificado é alterado para ser igual a $TV_i - 1$, onde 1 é determinado pela granularidade temporal da tabela. Assim, o registro anterior passa a ser histórico dando lugar aos valores do registro atual. A tabela 4 mostra o registro depois de uma operação de atualização de valores, considerando que a data atual seja Abril de 2005 e a granularidade seja de meses.

Fcodigo	Fnome	Fsalario	TVi	TVf
001	Roberto	2.600	Jan 2005	Mar 2005
001	Roberto	2.820	Abr 2005	<i>null</i>

Tabela 4: Atualização de um objeto

Na operação de **exclusão**, ocorre apenas uma exclusão lógica do registro mantendo-se seu histórico, assim o registro deixa de ser válido no tempo atual, passando a ser apenas parte do histórico deste objeto. Qualquer operação de exclusão altera o TV_f para um valor

igual a *now* - 1, onde *now* é o momento exato em que o registro está sendo excluído e 1 é determinado pela granularidade da tabela. É subtraída uma unidade de *now* indicando que o dado tornou-se parte do passado desse objeto. A tabela 5, mostra os dados após uma operação de exclusão, considerando que a data atual é Dezembro de 2005.

Fcodigo	Fnome	Fsalario	TVi	TVf
001	Roberto	2.600	Jan 2005	Mar 2005
001	Roberto	2.820	Abr 2005	Nov 2005

Tabela 5: Exclusão de um objeto

As operações de consulta também devem ser estendidas para que seja mais fácil efetuar consultas temporais. Por exemplo, para efetuar a seguinte consulta, "Listar os funcionários que estiveram empregados no período de Janeiro de 2005 até Dezembro de 2005", pode ser resolvido utilizando a sintaxe da linguagem TSQL2 exemplificada na figura 4.

```
SELECT nome FROM funcionario
WHERE VALID(funcionario) OVERLAPS PERIOD '[01/01/2005, 31/12/2005]'
```

Figura 4: Exemplo da operação de consulta temporal

2.2.3 Restrição de integridade

A especificação e monitoração de restrições de integridade são essenciais para qualquer aplicação de Banco de Dados (GERTZ; LIPECK, 1995), mantendo assim a consistência e confiabilidade dos dados. Esse conceito também se aplica aos Bancos de Dados Temporais, onde além de manter a integridade dos dados atuais (presente) também é necessário manter a integridade dos dados históricos e até mesmo dos dados futuros.

As restrições de integridade temporal possuem o componente dado e o componente temporal, onde esse limita a validade dos valores de cada atributo ao tempo de vida do objeto (TANSEL, 2004a; SOUZA, 1998). Por exemplo, os valores de salário dos funcionários do exemplo da tabela 1 não podem ter validade menor ou maior que o tempo de validade do próprio objeto funcionário. O componente temporal é peculiar aos dados temporais.

O componente temporal também restringe a existência do objeto (restrição de integridade temporal existencial), o relacionamento entre objetos (restrição de integridade temporal referencial) e restringe também a evolução dos dados ao longo do tempo.

No caso da restrição existencial, deve ser verificado se o período de validade de cada atributo ou conjunto de atributos que variam com o tempo esteja coerente com o período de validade do objeto ao qual pertençam. Considerando que TV_i e TV_f sejam respectivamente, o tempo de validade inicial e o tempo de validade final do objeto O e tvi e

tvf sejam respectivamente o tempo de validade inicial e o tempo de validade final de um atributo temporal A_t pertencente a esse objeto, então sempre deve ser verificado que:

- $TVi \leq tvi \wedge TVf \geq tvf$
- $TVi \leq TVf$

Na restrição existencial também deve ser verificado que não existam períodos sobrepostos para um mesmo objeto O ou para seus atributos temporais. Na tabela 6 observa-se que no período [Abr 2005, Jul 2005] existe uma inconsistência temporal, pois o funcionário apresenta dois salários.

Fcodigo	Fnome	Fsalario	TVi	TVf
001	Roberto	2.600	Jan 2005	Jul 2005
001	Roberto	2.820	Abr 2005	Dez 2005

Tabela 6: Restrição existencial - inconsistência temporal

No caso da restrição de integridade referencial temporal deve ser verificado que o período de validade do relacionamento entre dois objetos esteja contido na intersecção dos períodos de validade destes objetos. Considerando o relacionamento R entre os objetos O_1 e O_2 , então sempre deve ser verificado que:

- $[TVi^R, TVf^R] \supset [TVi^{O_1}, TVf^{O_1}] \cap [TVi^{O_2}, TVf^{O_2}]$

2.3 Trabalhos Realizados

Existem atualmente muitos trabalhos na área de sistema de banco de dados temporais, sempre explorando e se concentrando em três tópicos principais: (1) definição e descrição de um modelo temporal por um diagrama semelhante ao diagrama entidade relacionamento (DER), mas que consiga expressar a variação dos dados ao longo do tempo neste modelo; (2) linguagem de manipulação de dados, nesse caso muito trabalho foi desenvolvido por (SNODGRASS; KUCERA, 1995), esse e outros trabalhos de Richard Snodgrass (SNODGRASS et al., 1994) contribuíram para que o padrão da linguagem SQL incorporasse algumas operações temporais; e (3) implementação de modelos temporais, para que os atributos temporais, as restrições e a linguagem sejam definidos e utilizados em algum Sistema de Banco de Dados.

Outros trabalhos realizados da área influenciaram diretamente a definição de alguns conceitos e a formulação de idéias para o presente trabalho. Assim, apresentam-se abaixo alguns desses trabalhos:

O trabalho de (PINHEIRO, 2003) utiliza o pacote *Time Series Cartridge* implementado no Oracle à partir da versão 8i e também implementa o pacote de tempo de validade (PTV) para gerenciar o tempo de validade dos objetos manipulados no Sistema de Banco de Dados.

O pacote *Time Series Cartridge* é um conjunto de procedimentos, funções e objetos escritos em PL/SQL adaptados ao modelo relacional para gerenciar o armazenamento e recuperação de informações temporais relacionadas ao tempo de transação representado por um rótulo temporal (*timestamp*). O pacote de tempo de validade (PTV) desenvolvido pelo autor é implementado por meio de programação de gatilhos (*triggers*) e procedimentos (*procedures*) que adicionam algumas características funcionais para o gerenciamento temporal, tendo como principal objetivo manter a consistência entre os intervalos de tempo de validade.

Além de discutir e propor alguns conceitos sobre as operações físicas de remoção e atualização de dados, o autor tem como ponto central do trabalho as regras de consistência de dados considerando o tempo de validade, onde é especificado um conjunto de algoritmos que definem algumas regras para verificação e validação dos valores temporais dos dados que estão sendo manipulados. Por exemplo, se o período de validade de um objeto O ($TV(O)=[20, 50]$) que está sendo inserido no banco de dados se sobrepor de alguma forma a um outro registro ($TV(O)=[10, 30]$) deste mesmo objeto o Sistema de Banco de Dados deve garantir que essa sobreposição não ocorra de forma alguma, pois se isso ocorrer, na intersecção dos períodos haveria uma inconsistência, ou seja, no caso do objeto O o período $TV=[20, 30]$ teria dois valores válidos.

Algumas dessas regras, tornam possível que para que seja garantida a consistência, os períodos dos dados que estão sendo manipulados podem ter seus valores automaticamente modificados por algum procedimento programado internamente no SGBD. Apesar desse procedimento manter o Banco de Dados consistente em relação ao período válido, essas modificações automáticas dos períodos geram informações que podem não estar em conformidade com o mundo real. Ou seja, o SGBD não pode assegurar que o novo período que ele está atribuindo esteja em conformidade com o período do mundo real da aplicação.

A tabela 7 mostra uma inserção de dados inconsistentes por causa da sobreposição dos tempos de validade e a tabela 8 mostra uma possível correção efetuada por um procedimento de acordo com uma das regras propostas no trabalho. Nesse caso o próprio SGBD modificou o tempo de validade inicial do registro que foi inserido.

Nesse exemplo, em particular, observa-se que como havia uma sobreposição entre os intervalos $[2,4]$ e $[3,8]$ o SGBD alterou o segundo intervalo para $[5,8]$. Considera-se esta alteração "perigosa", pois é difícil garantir que a informação da 2ª tupla foi realmente válida no intervalo $[5,8]$. E também, será que a alteração da 1ª tupla para o intervalo

[2,3] e o da segunda para [4,8] não seria mais conveniente? Definir regras que determinam que o SGBD tome ações como as citadas pode não ser conveniente. Considera-se que esse tipo de inferência não é adequado. Além disso, se a cada dado inserido e/ou atualizado no sistema de banco de dados influenciar nos demais dados já previamente armazenados, o desempenho pode ser um fator limitante na questão técnica e funcional do produto (SGBDT).

codigo	nome	preco	TVi	TVf
123	Celular Nokia 6630	999,00	2	4
123	Celular Nokia 6630	699,00	3	8

Tabela 7: Dados inconsistentes

codigo	nome	preco	TVi	TVf
123	Celular Nokia 6630	999,00	2	4
123	Celular Nokia 6630	699,00	5	8

Tabela 8: Dados corrigidos

O trabalho de (OLIVEIRA, 2003) também explora as características do pacote *Time Series Cartridge* do Oracle 8i comparando com a nova funcionalidade do Oracle 9i, que absorveu muitos dos recursos do *Time Series Cartridge* na própria linguagem SQL, tornando a funcionalidade temporal mais integrada ao SGBD.

Nesse trabalho são efetuados alguns testes comparando as duas implementações de séries de tempo, onde na primeira implementação é utilizado o pacote *Time Series Cartridge* do Oracle 8i e na segunda implementação são utilizadas as funções incorporadas na linguagem SQL do Oracle 9i para tratar alguns aspectos de tempo. Um exemplo dessas funções absorvidas na linguagem SQL são as funções de agregação: soma (SUM), máximo (MAX), mínimo (MIN) e média (AVG) que foram estendidas para operarem sobre intervalos de tempo. Esses testes demonstram um maior desempenho em operações efetuadas utilizando a segunda implementação, além de mostrar a simplificação da sintaxe dos comandos.

A maior contribuição do trabalho foi a extensão do pacote PTV proposto em (PINHEIRO, 2003), acrescentando-lhe funcionalidade para ajudar na gerência do tempo de validade. Essas funções foram desenvolvidas em procedimentos do Oracle utilizando a PL/SQL. Um exemplo dessas funções desenvolvidas é: **tfirst(*tabela*, *tvi*, *tvf*)**, que procura a primeira ocorrência do atributo temporal *TVinicial* da *tabela* que satisfaça o critério de período passado pelo parâmetro da função, ou seja $TVinicial \geq tvi$ e $TVfinal \leq tvf$.

A implementação da funcionalidade temporal por meio de gatilhos e procedimentos nesses dois trabalhos anteriores tem a vantagem de incorporar diretamente no banco de dados tais funcionalidades, tornando-as transparentes para as aplicações.

Outro trabalho interessante que foi analisado (STEINER, 1998, 1999) tem como objetivo principal o desenvolvimento e implementação de uma linguagem temporal para manipular os aspectos temporais dos dados. Dessa maneira, em seu trabalho, o autor desenvolveu uma ferramenta para apoiar a implementação dessa linguagem, fornecendo um ambiente de teste para ajudar no seu refinamento.

A ferramenta, denominada *TimeDB*, oferece suporte à uma versão temporal da linguagem SQL chamada de ATSQL2 (SNODGRASS et al., 1996b, 1996a) e foi desenvolvida inicialmente (versão 1.0) na linguagem de programação C e posteriormente (versão 2.0) reescrita em JAVA para possibilitar a portabilidade para diferentes ambientes. Atualmente na versão 2.2, essa ferramenta tem suporte apenas para o tempo de validade, ao contrário da versão 1.0 que possuía a implementação bitemporal. Apesar disso, nessa última versão houveram melhorias na implementação da linguagem ATSQL2 onde foram incluídas funções de agregação temporal, restrições de chave primária e chave estrangeira, cláusula de agrupamento temporal (*group by*), além da funcionalidade anterior como consultas temporais, restrições temporais, tabelas e visões temporais. Algumas dessas características da linguagem estão apresentadas nos exemplos abaixo.

- Criação de uma tabela temporal. O exemplo da figura 5 mostra o comando de criação da tabela funcionário com três campos auto-explicativos, sendo que o campo `cod_funcionario` é a chave primária temporal desta tabela, ou seja, os valores no campo `cod_funcionario` devem ser únicos em cada intervalo de tempo. Além disso a tabela é especificada para ser temporal, assim, juntamente com cada tupla da tabela, é armazenado o tempo de validade.

```
CREATE TABLE funcionario (  
  cod_funcionario INT VALIDTIME PRIMARY KEY,  
  nome_funcionario VARCHAR(40),  
  salario NUMERIC(10,2)  
) AS VALIDTIME;
```

Figura 5: Criação de tabela temporal

- Inserção de dados na tabela. Considerando o exemplo anterior, é possível inserir dados na tabela utilizando uma extensão do comando INSERT implementada na ATSQL2. Observando os registros inseridos na figura 6, os intervalos são semi-abertos sendo que o instante inicial faz parte do período. A granularidade é anual,

assim, o primeiro registro representa o valor do salário do funcionário Rodrigo no período de 1995 até 1998, lembrando que o intervalo é aberto no instante final, ou seja, 1999 não pertence ao período. O último registro possui uma marcação especial (*forever*) determinando que este é o registro atual, ou seja, para o funcionário de código número 1, não há mais registros subsequentes.

```
VALIDTIME PERIODO [1995-1999)
INSERT INTO funcionario VALUES (1, 'Rodrigo Sakai', 1000.00);

VALIDTIME PERIODO [1999-2001)
INSERT INTO funcionario VALUES (1, 'Rodrigo Sakai', 1500.00);

VALIDTIME PERIODO [2001-2003)
INSERT INTO funcionario VALUES (1, 'Rodrigo Sakai', 2000.00);

VALIDTIME PERIODO [2003-forever)
INSERT INTO funcionario VALUES (1, 'Rodrigo Sakai', 2500.00);
```

Figura 6: Inserção de dados temporais

- Restrição de chave primária temporal. Como especificado na criação da tabela, o campo `cod_funcionario` é uma chave primária temporal. Diferentemente de uma chave primária não-temporal onde o valor da chave deve ser único na tabela inteira, o valor de uma chave primária temporal deve ser único apenas em um mesmo período. Assim, o exemplo da figura 7 resultaria em uma violação da chave primária, pois este período se sobrepõe aos períodos de dois outros registros. No caso específico da ferramenta *TimeDB*, o registro que viola a restrição é inserido normalmente no Banco de Dados e não é mostrado nenhum erro ou aviso, mas quando se executa uma operação de seleção, os registros que estiverem violando esse tipo de restrição é descartado do resultado.

```
VALIDTIME PERIODO [1999-2002)
INSERT INTO funcionario VALUES (1, 'Rodrigo Sakai', 2800.00);
```

Figura 7: Restrição temporal

- Seleção de registros. A operação de seleção pode ser efetuada recuperando apenas os registros atuais, ou seja, os registros que ainda são válidos, pode recuperar todo o histórico de um objeto ou pode recuperar os dados de determinado período. Essas operações são mostradas respectivamente na figura 8.

No entanto, o *TimeDB* não faz parte internamente de um SGBD, ele é uma API que basicamente traduz as instruções realizadas em ATSQL2 (SQL Temporal) para a


```
SELECT * FROM funcionario WHERE nome = 'Rodrigo Sakai';  
  
VALIDTIME  
SELECT * FROM funcionario WHERE nome = 'Rodrigo Sakai';  
  
VALIDTIME PERIOD[1999-2003)  
SELECT * FROM funcionario WHERE nome = 'Rodrigo Sakai';
```

Figura 8: Seleção temporal

SQL padrão, onde então essa instrução será executada em um SGBD não temporal. A comunicação entre a ferramenta e o SGBD é realizada por meio de *drivers* no padrão JDBC, o que possibilita a utilização de diferentes Servidores de Banco de Dados como Oracle, Sybase, Cloudscape entre outros.

Além da implementação da linguagem ATSQL2, o *TimeDB* oferece uma interface gráfica simples que permite o usuário executar as instruções da linguagem como demonstrada nos exemplos anteriores, e também possui um conjunto de classes que permite o desenvolvimento de aplicações JAVA que precisam executar instruções temporais.

As informações temporais são armazenadas em metadados (tabelas) e campos adicionais criados pela própria ferramenta no SGBD alvo, por exemplo, a tabela criada anteriormente produz na verdade uma tabela no SGBD com mais dois campos, em que serão armazenados os valores dos tempos de validade inicial e final representados em segundos.

Com esta estrutura independente, torna-se possível manter aplicações não temporais que acessam diretamente o SGBD e aplicações temporais que acessam o SGBD por meio do *TimeDB*.

Nesse trabalho, a implementação e utilização de uma linguagem temporal (ATSQL2) mostrou para os pesquisadores envolvidos que a maior dificuldade não está em armazenar os períodos de tempo válido, mas efetuar consultas, cálculos e atualizações históricas e ainda manter a consistência da informação (STEINER, 1999). A vantagem da arquitetura implementada é que ela pode ser utilizada para diferentes SGBDs e ainda retira a responsabilidade do desenvolvedor de ter que criar aplicativos que mantenham a consistência temporal.

3 GERÊNCIA DAS INFORMAÇÕES TEMPORAIS

No trabalho de (SOUZA, 1998) são especificados: um modelo bitemporal orientado a objetos, as restrições de integridade pertinentes a esse modelo e uma linguagem para manipulá-lo. Esse conjunto de conceitos e definições serviram para formar a base deste trabalho. O modelo especificado em (SOUZA, 1998) utiliza as duas classes de tempo (tempo de validade e tempo de transação) descritas no capítulo anterior na seção 2.1 e utiliza também o paradigma de orientação a objetos que é mais apropriado que o modelo relacional, pois permite a representação mais natural do atributo temporal.

O presente trabalho está considerando apenas o tempo de validade, pois essa é a classe temporal que está intimamente ligada ao contexto da aplicação, sendo mais susceptível às restrições de integridade impostas pelo ambiente.

3.1 Introdução

As restrições de integridade de dados de um Sistema de Banco de Dados (SBD) têm como objetivo assegurar que as operações de atualização (inclusão, remoção e modificação) de dados realizadas no SGBD resultem em informações sempre corretas com relação ao domínio da aplicação. Ou seja, os valores atribuídos aos campos devem estar de acordo com a realidade da aplicação. Por exemplo, se for definido que um funcionário não pode possuir o salário negativo, qualquer operação que infrinja essa restrição deve ser rejeitada.

As restrições de integridade de dados definidas em um sistema fazem parte dos requisitos funcionais originados pelas regras de negócio que regem a aplicação. Essas regras são implementadas no sistema para permitir ou não que as operações sejam efetuadas e produzam alterações no Banco de Dados. Essas restrições podem ser consideradas como a expressão formal de uma regra de negócio da aplicação no sentido de manter os dados coerentes com seu domínio.

Especificar e monitorar essas restrições de integridade é essencial para qualquer tipo de aplicação de Banco de Dados, pois ajuda a garantir a corretude e confiabilidade dos dados (GERTZ; LIPECK, 1995).

A adição dos aspectos temporais nos SGBDs traz a necessidade de incorporar e es-

tender as restrições de integridade com a finalidade de continuar mantendo a consistência da informação.

Para incorporação desses aspectos temporais no SGBD, um novo tipo de dado é proposto conforme descrito na seção 3.2. Este novo tipo de dado e as operações a ele pertinentes auxiliam na manipulação dos dados temporais e na definição dessas restrições.

3.2 O Tipo de Dado "t_perodo"

O intervalo temporal possui uma semântica própria e diferente dos outros tipos de dados no contexto do SGBD. Ele é delimitado por dois valores que determinam o ponto inicial e o ponto final da validade de uma informação. É possível representar o intervalo temporal por meio de marcas temporais que armazenam o valor do tempo de validade inicial (TV_i) e o valor do tempo de validade final (TV_f) de um objeto.

A figura 9 exemplifica essas marcas temporais, onde $T_validadeI$ representa o tempo de validade inicial e $T_validadeF$ representa o tempo de validade final do objeto funcionário.

Figura 9: Classe temporal funcionário

Na figura 9, as duas marcas temporais são utilizadas para armazenar o intervalo de tempo de validade para esse objeto. Nesse caso, é necessário implementar uma regra que sempre garanta que os valores atribuídos a essas marcas obedçam a sentença $TV_i \leq TV_f$, ou seja, o tempo de validade inicial de qualquer objeto deve sempre ser anterior ou igual ao tempo de validade final desse mesmo objeto.

A regra principal ($TV_i \leq TV_f$) sobre o intervalo temporal garante que uma informação sempre possua um início temporal (TV_i) conhecido e um final temporal (TV_f) igual ou posterior ao valor do seu início. O TV_f pode ser desconhecido, indicando que o intervalo de validade desse objeto ainda não terminou e/ou não tem um final definido.

Os valores registrados para TV_i e TV_f devem ser definidos pelo usuário. Dependendo dos valores definidos a regra $TV_i \leq TV_f$ pode ser infringida. Em particular, para os

valores especiais *null* (indefinido) e *now* (corresponde ao valor de tempo corrente) os seguintes casos podem resultar na violação dessa regra (SOUZA, 1998):

1. $TV_i = null$ e $TV_f < now$ Caso o usuário não atribua valor para TV_i , o SGBD atribuirá o valor de *now* para essa marca de tempo. Nesse caso, como o valor fornecido para TV_f é menor que o valor do tempo corrente, ocorreria a violação de $TV_i \leq TV_f$.
2. $TV_i = now$ e $TV_f < now$ De acordo com o que foi exposto anteriormente, também ocorreria a violação de $TV_i \leq TV_f$.
3. $TV_i < now$ e $TV_f < now$ e $TV_i > TV_f$ Os valores atribuídos para TV_i e TV_f podem ser referentes ao tempo passado, desde que não viole $TV_i \leq TV_f$.
4. $TV_i > now$ e $TV_f > now$ e $TV_i > TV_f$ Os valores atribuídos a TV_i e TV_f podem ser referentes ao tempo futuro, desde que não viole $TV_i \leq TV_f$.

Nestes casos, onde são utilizados dois atributos independentes para a representação do intervalo temporal, como mostrado na figura 9, todas essas condições sempre devem ser verificadas e implementadas em objetos temporais que utilizam essas marcas (TV_i e TV_f). Quando isso acontece, essa responsabilidade é direcionada para o sistema que deve prover métodos (funcionalidade) que sempre verifiquem essas regras para todos os objetos temporais criados. Desta forma, os desenvolvedores ficam mais sobrecarregado, pois são eles que devem implementar esses métodos em cada componente do sistema que manipule as informações temporais.

Neste trabalho, para permitir a gerência adequada dos valores atribuídos a essas marcas de tempo e uma manipulação mais fácil por parte do usuário, propõe-se um novo tipo de dado. Esse tipo de dado é constituído por dois campos que representam os limites inferiores e superiores do intervalo temporal, respectivamente representados por TV_i e TV_f . O novo tipo de dado proposto foi denominado de **t_periodo** e sua estrutura está representada na figura 10.

O tipo de dado *t_periodo* apresenta uma estrutura atômica que representa o intervalo temporal. Essa estrutura é considerada atômica, pois tanto o valor de TV_i quanto o valor de TV_f devem ser manipulados conjuntamente, de forma que esses valores sejam armazenados sob uma única referência. Não é permitido que somente um dos valores seja, por exemplo, inserido na tabela. Isso impede que o desenvolvedor esqueça de inserir o valor para todo o período.

Esse novo tipo de dado é utilizado para substituir os dois atributos anteriores (TV_i e TV_f) que representavam as marcas temporais dos objetos. Seguindo o mesmo exemplo

```

struct t_perodo {
    date TVi;
    date TVf;
}

```

Figura 10: Estrutura do tipo de dado *t_perodo*

da classe temporal funcionário, é possível criar um objeto que utilize um único atributo do tipo de dado *t_perodo* para representar o intervalo temporal. A figura 11 apresenta um exemplo utilizando o novo tipo de dado *t_perodo*.

Figura 11: Classe temporal funcionário com o tipo de dado *t_perodo*

Comparando os atributos da classe criada na figura 9 com os atributos da classe da figura 11 nota-se que os atributos *T_validadeI* e *T_validadeF* do tipo de dado *Date* foram substituídos por um único atributo *T_validade* do tipo *t_perodo*.

Assim, é possível representar de forma consistente a informação temporal utilizando-se a primitiva de intervalo temporal apresentada na seção 2.1.5. Os valores temporais ficam consistentes, pois o tipo de dado *t_perodo* garante que as regras definidas para o intervalo temporal sejam obedecidas.

Como qualquer outro tipo de dado o *t_perodo* é definido por um par (V, O) , onde *V* representa o domínio de valores que este tipo pode assumir e *O* representa as operações para esse tipo de dado.

O tipo de dado *t_perodo* foi criado para assumir um valor que representa um intervalo temporal. Cada um dos dois elementos da estrutura do *t_perodo* pode então assumir qualquer valor cujo domínio seja temporal (ano, mês, dia, hora, minuto, segundo, etc). Isso constitui o domínio (*V*) dos valores para esse tipo de dado.

Similar a outros tipos de dados um conjunto de operações e operadores devem estar disponíveis para o tipo *t_perodo*. Considerando as peculiaridades associadas ao registro do tempo no BD um conjunto de operadores binários (ALLEN, 1983; DATE, 2004) devem estar disponíveis para os usuários. Esses operadores permitem por exemplo, verificar se um intervalo é igual a outro ou se eles se sobrepõem. Esses operadores são detalhados na seção 3.2.1.

3.2.1 Operadores de Manipulação para *t_perodo*

As marcas temporais representadas por TV_i e TV_f podem ser manipuladas pelas operações já existentes em qualquer SGBD. Os operadores relacionais ($\leq$, $<$, $\geq$, $>$, $=$) podem ser utilizados para comparar cada marca temporal com um valor de tempo especificado. Por exemplo, para verificar quais funcionários (cuja estrutura está na figura 9) existem na empresa desde Janeiro de 2006, deve-se utilizar a condição de seleção **T_validadeI $\geq$ 'Janeiro-2006'** na consulta SQL para comparar a marca temporal do objeto funcionário com um valor de tempo, e assim, obter o resultado desejado.

Também é possível delimitar em uma consulta os valores das duas marcas temporais (TV_i e TV_f) como demonstrado na figura 12. Nessa figura, o exemplo mostra uma consulta efetuada para verificar quais os funcionários que trabalharam na empresa no período de Janeiro até Julho de 2006, ou seja, começaram a trabalhar em janeiro de 2006 e por algum motivo não trabalham mais na empresa desde julho de 2006.

```
SELECT * FROM funcionario
WHERE T_validadeI >= '01-jan-2006'
and T_validadeF <= '31-jul-2006';
```

Figura 12: Operação de seleção com operadores relacionais

O problema com essa solução para inclusão e manipulação do dado temporal é que o usuário/aplicação é o responsável por definir atributos relativos as marcas de tempo ($T_validadeI$ e $T_validadeF$ da figura 9) para o registro da primitiva temporal intervalo.

Obviamente que o usuário sempre será responsável por definir qual dado é temporal ou não, pois é um requisito da aplicação. Porém, entende-se que o SGBD deve fornecer mecanismos apropriados para representação do dado temporal, levando em consideração a primitiva temporal usada para o registro do tempo.

Os operadores relacionais, lógicos e aritméticos existentes no SGBD não podem ser utilizados diretamente para manipular o tipo de dado *t_perodo*, pois ainda não foram implementados para trabalhar com essa nova estrutura. Para isso, é necessária a implementação desses operadores utilizando o tipo de dado *t_perodo*. Isso pode ser realizado devido a sobrecarga de operadores, que permite a sua redefinição para que seja possível aplicar as operações em diferentes estruturas. Por exemplo, para somar dois períodos distintos que se sobrepõe (overlaps) pode ser utilizado o sinal de adição (+), desde que esse operador aritmético seja implementado para efetuar operações sobre o tipo de dado *t_perodo*. O resultado dessa soma seria um intervalo de tempo onde TV_i corresponderia ao menor valor das marcas temporais e TV_f corresponderia ao maior valor das marcas temporais entre esses dois intervalos.

Além da sobrecarga dos operadores que já existem no SGBD, o novo tipo de dado *t_periodo* precisa que um conjunto diferente de operações seja implementado permitindo a manipulação dos valores armazenados nessa nova estrutura. Esse novo conjunto de operações é uma implementação dos operadores binários propostos por (ALLEN, 1983; DATE, 2004). Esses operadores podem ser utilizados para manipular os intervalos temporais efetuando comparações entre eles.

Operador	Descrição	Exemplo
Equal	Verifica se os períodos são exatamente iguais
Before	Verifica se um período vem após o outro
Meets	Verifica se um período se encontra com o outro período
Overlaps	Verifica se um período se sobrepõe a outro período
During	Verifica se a ocorrência de um período ocorre durante a ocorrência de outro período
Starts	Verifica se o tempo de validade inicial (TVi) entre dois períodos são iguais
Finishes	Verifica se o tempo de validade final (TVf) entre dois períodos são iguais

Figura 13: Operadores de Allen

A figura 13 representa esses sete novos operadores. Na coluna "**exemplo**", cada barra representa intervalos de tempo de dois objetos que estão sendo comparados. A extremidade esquerda de cada barra representa o tempo de validade inicial (TV_i) para aquele período e consequentemente a extremidade direita de cada barra representa o tempo de validade final (TV_f) do mesmo período que está sendo representado. O eixo temporal que tem ordenação linear (seção 2.1.2), é representado pela seta, que indica a passagem do tempo considerando o início no instante 0 (zero) e o fim infinito (desconhecido).

Para exemplificar a utilização desse novo conjunto de operadores foi efetuada a mesma operação da figura 12, utilizando, desta vez, o novo tipo de dado *t_periodo* e o novo operador **OVERLAPS** como demonstrado na figura 14. A classe funcionário está apresentada na figura 11.

A consulta realizada na figura 14 compara um período (['01-jan-2006', '31-jul-2006']) com a marca temporal (T_validade) do objeto funcionário. Nessa consulta, os valores do período representam os limites inferiores e superiores que devem ser verificados.

```
SELECT * FROM funcionario  
WHERE T_validade OVERLAPS ['01-jan-2006','31-jul-2006'];
```

Figura 14: Operação de seleção com o operador *overlaps*

3.3 Restrições de Integridade Temporal

Como descrito no início deste capítulo, as restrições de integridade fazem parte dos requisitos funcionais de uma aplicação. Elas podem implementar parte das regras de negócio no sentido de manter a consistência da base de dados e a coerência da informação com o mundo real.

As restrições de integridade temporal são aplicadas somente ao contexto dos objetos temporais. Os objetos que não apresentam a característica temporal são afetados somente pelas restrições não temporais dos SGBDs (SOUZA, 1998).

Na grande maioria dos SGBDs é possível criar essas restrições no próprio repositório, centralizando desta forma a verificação da consistência de todas as informações.

Com a adição de uma nova estrutura (tipo de dado *t_periódico*) no Banco de Dados é necessário que a integridade e consistência das informações sejam mantidas também nessa estrutura.

3.3.1 Chave Primária Temporal

O modelo relacional propõe que cada objeto de um conjunto deve ser distinto dos demais objetos do mesmo conjunto. Para isso, no caso dos Sistemas Gerenciadores de Banco de Dados Objeto-Relacionais (SGBDORs), todos os objetos criados recebem automaticamente um número que é único em todo o BD. Esse número é denominado de *Object Identifier* (OID) e é utilizado para distinguir univocamente um objeto dentro de um conjunto.

Geralmente o OID é um número inteiro sequencial e é gerado automaticamente pelo próprio SGBD quando o objeto é criado (instanciado). Por exemplo, quando inserimos um novo funcionário na classe apresentada na figura 11 o SGBD se encarrega de atribuir automaticamente a esse objeto da classe funcionário um valor exclusivo de OID. Esse OID vai pertencer eternamente ao objeto. Mesmo que ele seja removido do BD, nenhum outro objeto irá utilizar o seu OID.

O OID poderia ser utilizado então como chave primária das classes não-temporais ou temporais. Mas, observa-se que este identificador é gerado automaticamente pelo SGBD e não tem nenhum significado importante para a realidade em que a aplicação está inserida. Por exemplo, o SGBDOR pode ter gerado o número 8765 para o objeto

funcionário anterior. Percebe-se que esse número não significa absolutamente nada para a aplicação e seus usuários, não sendo utilizado na chave de busca em consultas definidas pela aplicação.

Desta forma, adota-se o conceito de chave primária oriunda do modelo relacional. A chave primária corresponde a um subconjunto de atributos de uma classe que permite identificar um único objeto dessa classe.

No caso dos BDTs, o conceito de chave primária deve ser estendido para que as classes temporais possuam um conjunto de atributos que identifiquem unicamente cada objeto pertencente a ela.

A chave primária de uma classe temporal não pode ser constituída apenas pelos atributos não temporais, pois eles não são suficientes para garantir a existência de um objeto em um único intervalo de tempo. Na figura 15, apresenta-se dois objetos temporais.

Observa-se na figura 15 que o objeto "O1" existe em três períodos e que há intervalos em que o objeto não tem existência lógica para a aplicação. Neste exemplo, pode-se pensar no caso do funcionário que foi contratado três vezes pela mesma empresa em períodos diferentes e que houve um certo momento em que ele efetivamente não trabalhou nessa empresa.

Ainda observa-se na figura 15 que para o objeto "O2" ocorre uma sobreposição de intervalos de tempo. Essa sobreposição leva a inconsistência na base de dados, pois fica indeterminada a validade do objeto no intervalo correspondente a sobreposição.

Por exemplo, o funcionário "O2" não pode ser contratado duas vezes ao mesmo tempo pela mesma empresa, primeiro no intervalo $[t_1, t_3]$ e depois no intervalo $[t_2, t_4]$, levando a sobreposição (*overlaps*) mostrada a seguir.

$$[t_1, t_3] \cap [t_2, t_4] = t_2^1 \text{ ou}$$

$$[t_1, t_3] \cap [t_2, t_4] = [t_2, t_3]^2$$

Isto implicaria que o empregado fosse recontratado estando ainda contratado.

Para evitar essa ocorrência, a chave primária temporal deve ser constituída por atributos não temporais e mais uma marca de tempo. No caso, pode ser utilizado o tempo de validade para formar essa chave primária.

O trabalho de (TANSEL, 2004a) utiliza um atributo não-temporal e mais o tempo de validade inicial para compor a chave primária. Observando-se novamente a figura 15 é possível notar que no caso do objeto 2 ("O2") os tempos de validade inicial dos dois períodos são diferentes e mesmo assim existe a sobreposição dos intervalos temporais,

¹Considerando a linha de tempo discreta e ordenada

²Considerando a linha de tempo contínua e ordenada

Figura 15: Tempo de vida dos objetos

ou seja, os dados continuam inconsistentes. O mesmo aconteceria caso fosse utilizado somente o tempo de validade final do objeto para compor a chave primária.

Então, nota-se que para alguns tipos de aplicações temporais é necessário considerar a utilização dos dois atributos (TV_i e TV_f).

No trabalho proposto emprega-se um único atributo temporal ($t_periodo$), então para que não haja inconsistências como a do objeto O2 (figura 15), a chave primária deve ser constituída pelo valor de um atributo chave mais o atributo $t_periodo$.

Assim, tomando como exemplo a figura 9 que representa o modelo da classe Funcionario, é possível definir a chave primária para esta classe como sendo constituída por: *numeroFuncionario* e $t_periodo$.

3.4 Especificação das Restrições de Integridade Temporal

A definição e implementação do tipo de dado $t_periodo$ auxilia na posterior implementação das restrições de integridade temporal. O próprio tipo de dado garante que o tempo de validade inicial seja menor ou igual ao tempo de validade final, ou seja que a regra $TV_i \leq TV_f$ seja sempre respeitada pelas operações.

Além disso, quando um objeto temporal possui atributos que variam no tempo, deve ser verificado se o período de validade de cada atributo temporal está dentro do período de vida do objeto (SOUZA, 1998) conforme especificado no capítulo anterior na seção 2.2.3.

3.4.1 Restrição de Integridade Existencial Temporal

A restrição de integridade existencial mantém a consistência de cada objeto de acordo com a chave primária. Assim, dada uma classe temporal C e um objeto c_i dessa classe, a integridade existencial garante basicamente que:

1. O valor de cada atributo que compõe a chave primária do c_i pertencente a um conjunto C deve ser diferente de nulo. Isto ocorre porque um atributo nulo impede a comparação entre os valores dos objetos dessa classe, tornando impossível determinar um objeto único dentro dela.

Assim, uma marca de tempo que componha a chave primária de uma classe temporal também não pode ser nula. Pode ocorrer um intervalo de tempo nulo quando:

- $TV_i = null$ e $TV_f = null$;
- $TV_i = null$ e $TV_f \neq null$.

No primeiro caso, não foi atribuído nenhum valor para o tempo de validade inicial do objeto nem para o tempo de validade final do objeto. No segundo caso, foi atribuído um valor diferente de nulo para o tempo de validade final, mas não foi atribuído nenhum valor para o tempo de validade inicial do objeto.

Como especificado no início desta seção, o próprio tipo de dado temporal verifica se um de seus atributos é nulo. Assim, esta consistência é garantida em qualquer objeto temporal que utilize o tempo de validade definido como $t_periodo$.

2. Não deve existir outro objeto c_j em C com o mesmo valor da chave primária de c_i . Isto garante que um objeto consiga ser selecionado e manipulado de maneira única no sistema. No caso de uma classe temporal, onde a marca de tempo faz parte da chave primária, é necessário garantir também que não ocorra nenhuma sobreposição dos valores temporais ($t_periodo$) ocasionando a existência de valores inconsistentes.

Essa regra foi implementada com o tipo de dado $t_periodo$, onde deve ser verificada a ocorrência de períodos que se interseccionam considerando o mesmo objeto. Caso exista, essa operação é rejeitada pelo SGBD.

Nesta classe de restrição temporal também é necessário verificar o tempo de validade dos atributos temporais (tv) com relação ao tempo de validade do próprio objeto (TV). Ou seja, é necessário verificar se o tempo de validade dos atributos temporais estão contidos no tempo de vida desse objeto

Para implementar essa restrição utilizando o tipo de dado $t_periodo$ pode ser definido que se o tv (tempo de validade do atributo temporal) não estiver contido em TV (tempo de validade do objeto temporal) então a manipulação dos valores dos atributos temporais deve ser rejeitada. No exemplo da figura 16 deve ser verificado se $salario.t_validade$ *DURING* $funcionario.t_validade$, caso não seja verdadeiro, a operação é rejeitada.

Figura 16: Objeto funcionario com o atributo temporal salário

3.4.2 Restrição de Integridade Referencial Temporal

A integridade referencial mantém a consistência no relacionamento entre dois objetos distintos, onde os valores de um desses objetos referenciam valores existentes do outro objeto.

Assim, dada duas classes temporais distintas C e D e seus respectivos objetos c_i e d_i . Considerando a chave primária de c_i como $PK(c_i)$, a integridade referencial é mantida por meio de um conjunto de atributos de d_i que referencia a chave primária de c_i . Esse conjunto de atributos de d_i é chamado de chave estrangeira.

Essa definição implica que os atributos que constituem uma chave estrangeira satisfaçam as duas regras seguintes:

1. Os atributos da chave estrangeira têm os mesmos domínios da chave primária.

No caso de classes temporais pode surgir uma nova condição de domínio, onde o intervalo temporal de d_i deve ter granularidade igual ou menor que o intervalo temporal de c_i . Isto deve ocorrer para que seja possível associar os dois objetos.

2. Os valores da chave estrangeira de d_i possuem os mesmos valores da chave primária da classe que é referenciada, no caso c_i .

Para as classes temporais é necessário descrever três situações diferentes:

- Um objeto temporal referenciando um outro objeto temporal. Neste caso, o relacionamento entre esses objetos é válido na intersecção dos períodos de cada um. Por exemplo, considerando os dois objetos temporais gerente e funcionário, o relacionamento temporal entre eles ocorre no período de intersecção determinando quando cada funcionário foi gerenciado por determinado gerente.
- Um objeto não temporal referenciando um objeto temporal. Neste caso, para esse tipo de relacionamento considera-se apenas os atributos não temporais da chave primária do objeto temporal. Isso significa que o objeto não temporal pode referenciar o objeto temporal em qualquer intervalo de tempo. O relacionamento é finalizado com a remoção do objeto não temporal ou a finalização do período de validade do objeto temporal, que pode ocorrer naturalmente com o término do período ou por alguma atualização ou remoção do objeto temporal.

- Um objeto temporal referenciando um objeto não temporal. Da mesma forma como no item anterior, é considerado apenas os atributos não temporais para efetuar a consistência referencial. Por exemplo, considerando o objeto não temporal departamento e o objeto temporal funcionário, o relacionamento existente entre eles é independente da marca de tempo válido. Semelhante ao item anterior, o relacionamento é finalizado pela remoção do objeto não temporal ou pela finalização do período de validade.

3.5 Operações de Manipulação de Dados e Violação de Restrições de Integridade

Considerando as restrições de integridade temporal descritas na seção anterior é necessário definir as operações de atualização que podem violar as restrições implementadas no SGBD.

É considerada como operação de atualização qualquer operação que modifique o estado de um Banco de Dados. Ou seja, a inclusão de uma nova informação (*INSERT*), a remoção (*DELETE*) ou a atualização (*UPDATE*) de uma informação existente.

A operação de consulta (*SELECT*) a um Banco de Dados não modifica o seu estado, portanto não é considerada como operação de atualização e não pode violar nenhuma restrição de integridade.

Essas atualizações devem ocorrer de maneira que após sua execução o novo estado do Banco de Dados continue sendo consistente. Caso ocorra a violação de alguma restrição de integridade o SGBD deve rejeitar a operação que estiver sendo executada.

3.5.1 Operação de Inserção

A operação de inserção pode violar a **restrição de domínio** quando o valor que está sendo atribuído não corresponde ao domínio especificado para o atributo. No caso da marca temporal, o intervalo de tempo representado pelos valores de TV_i e TV_f deve respeitar a regra $TV_i \leq TV_f$ e a granularidade conferida.

Se o valor da chave primária de um objeto já existir, a operação também não poderá ser realizada, pois estará violando a **restrição de chave**. Para os objetos temporais, essa restrição garante que não existam valores duplicados para os atributos temporais, onde os períodos se sobreponham em algum momento. Por exemplo, na figura 16, um funcionário pode ter vários salários desde que cada um seja em um período diferente.

Para garantir a **integridade de entidade**, os valores atribuídos para a chave primária do objeto têm que ser diferentes de nulo. Pela própria definição da chave primária,

qualquer campo que faça parte da chave primária não pode ser nulo, assim também, as marcas temporais TV_i e TV_f representadas pelo $t_periodo$ não podem ser nulas.

Além disso, considerando um objeto temporal, o período de validade de cada atributo temporal pertencente a esse objeto deve estar contido em seu período de validade. Por exemplo, o atributo temporal salário da figura 16 deve ter os valores de sua marca temporal ($t_validade$) contida no período de validade do objeto funcionário. A integridade de entidade também é conhecida como integridade existencial.

Para garantir a **integridade referencial temporal**, o relacionamento que existe entre dois objetos temporais deve ter algum intervalo de sobreposição. A figura 17 mostra dois objetos temporais, departamento e funcionário. A atributo salário foi suprimido da figura para simplificar o exemplo.

Figura 17: Relacionamento entre o objeto departamento e funcionário

Por exemplo, considerando o departamento de contabilidade que teve seu funcionamento de 20 de janeiro de 1990 até 10 de julho de 2001 e um funcionário recém contratado, que tem seu início de trabalho marcado em 15 de junho de 2006. O SGBD deve garantir que não se efetivem operações que tentem relacionar esses dois objetos, pois não há um período em que eles possam co-existir.

3.5.2 Operação de Remoção e Operação de Atualização

A operação de remoção pode infringir a integridade referencial ou a integridade existencial temporal. A primeira, ocorre quando o objeto que está sendo removido continua participando de um relacionamento com no mínimo um outro objeto.

Na figura 17, não é possível remover um departamento que esteja sendo referenciado por um funcionário, pois se isso ocorresse, existiriam funcionários que não estão alocados em departamentos em alguns momentos, mesmo que esses funcionários existam na empresa.

A operação de atualização, quando utilizada para modificar os valores de atributos que não façam parte da chave primária, não pode infringir nenhuma restrição, além da restrição de domínio. Ou seja, se for atualizar uma marca temporal, é necessário garantir que os novos valores obedeçam a regra $TV_i \leq TV_f$.

Porém, modificar o valor da chave primária tem o mesmo efeito que apagar um objeto

e inserir outro em seu lugar, sendo então que as restrições para a inserção e remoção de objetos podem ser aplicadas para a operação de atualização.

Na atualização, se os valores da chave estrangeira forem modificados, o SGBD tem que continuar garantindo a integridade referencial e verificar se o valor modificado existe nos relacionamentos desse objeto.

4 EXTENSÃO DAS FUNCIONALIDADES DO SGBD PARA CONTROLE DA VARIAÇÃO TEMPORAL

Este capítulo apresenta uma proposta de implementação de parte dos aspectos temporais apresentados no capítulo 3. O tipo de dado temporal, seus operadores e as restrições de integridade são adicionados ao contexto do SGBD utilizando-se um módulo temporal.

A arquitetura do SGBD é descrita nesse capítulo de maneira a mostrar onde o módulo temporal pode ser introduzido no contexto do SGBD.

4.1 Introdução

No passado, os Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDRs) ofereciam pouco ou nenhum suporte que tornasse possível a extensão de sua funcionalidade. Com essa rigidez, era inviável a implementação de novas funções dentro do SGBD sem modificar boa parte de seu código. Desta maneira, era necessário efetuar uma grande modificação na linguagem SQL e na arquitetura dos SGBDs para adicionar a funcionalidade temporal (CHEN; KONG; ZANIOLO, 2003) ou qualquer outra funcionalidade diferente das implementadas.

Com a demanda do mercado, um novo conceito de SGBD foi proposto de maneira a oferecer aos desenvolvedores uma maior flexibilidade de implementação. Os Sistemas Gerenciadores de Banco de Dados Objeto-Relacionais (SGBDOR) oferecem a possibilidade de extensibilidade, que apesar de ainda ter algumas limitações, pode ser utilizada para agregar funcionalidade temporal em um SGBDOR.

Dessa forma, um SGBD objeto-relacional foi escolhido para implementar alguns aspectos temporais, sendo possível utilizar as vantagens de sua funcionalidade, destacando-se entre elas:

- **Extensibilidade:** permite a definição de novas estruturas de armazenamento, novos tipos de dados e novos operadores para manipular esses tipos de dados. Essa característica ajuda a implementar um novo tipo de dado para armazenar o tempo de validade dos objetos e também implementar as operações sobre este tipo de dados.

- **Objetos complexos:** são objetos compostos por outros objetos. Essa característica permite aninhar objetos dentro de objetos, por exemplo o atributo temporal salário se torna um objeto aninhado dentro do objeto funcionário.
- **Linguagem de manipulação:** neste tipo de SGBD continua sendo implementado o padrão SQL. Além dessa linguagem ser bem difundida e utilizada, ainda permite que as aplicações anteriores possam usar o SGBD sem haver necessidade de modificação da aplicação ou código SQL.

Além disso, a Linguagem de manipulação de dados SQL3 (ISO/IEC, 1999) implementada nesses SGBDs traz recursos para manipular estruturas como tabelas aninhadas (objetos complexos) e atributos do tipo *array* entre outros que são úteis para a proposta deste trabalho.

4.2 Arquitetura do SGBD Temporal

Esta seção descreve de maneira sucinta os componentes que constituem a arquitetura padrão de um SGBD, destacando-se também a interação entre esses componentes.

4.2.1 Componentes de um SGBD

A arquitetura típica da maioria dos SGBDs é definida por um conjunto de estruturas e métodos agrupados em componentes, onde cada um tem seu conjunto de operações e interfaces bem definidas, proporcionando um baixo acoplamento entre eles. Isso permite que algumas modificações em um dos componentes não exija nenhuma ou muito pouca modificação nos demais.

A figura 18 mostra estes componentes e a interação entre eles. A função de cada um desses componentes é descrita a seguir:

- **Processador de Consulta** - Responsável por receber as requisições de consultas realizadas pelos aplicativos clientes. O processamento de consultas é uma operação complexa; sub-componentes executam cada parte da tarefa. O *parser* recebe a instrução, valida a sintaxe e converte a instrução SQL em uma árvore de busca. O reescritor (*rewriter*) recebe esta árvore de busca e se necessário realiza transformações nela a fim de que seja utilizada pelo otimizador. O otimizador recebe os resultados do reescritor e cria todas as possibilidades para executar a consulta, avalia as possibilidades e escolhe o melhor plano de execução. E por fim, o *executor* efetiva o comando no SGBD retornando ao aplicativo uma resposta com a informação desejada.

Figura 18: Arquitetura do SGBD PostgreSQL

- **Gerenciamento de Arquivos e Métodos de Acesso** - Oferece métodos de acesso aos dados, ou seja, algoritmos de busca e indexação de dados, como por exemplo *heap*, *hash*, *btree* e *rtree*. Esse componente decide qual o melhor método a ser utilizado para o acesso a uma determinada informação utilizando para isso o plano de execução criado na fase anterior e as estatísticas de acesso aos dados. A definição e manipulação do catálogo global, onde estão contidas as informações sobre o próprio sistema, também é responsabilidade desse componente.
- **Gerenciamento do Buffer** - Responsável pelo gerenciamento da área de trabalho do SGBD. Todas as operações de leitura e escrita ocorrem primeiro no buffer para depois serem efetivadas fisicamente no BD. Este componente gerencia da melhor maneira possível o espaço de memória reservada ao SGBD, diminuindo assim o acesso físico ao disco. Por exemplo, ele mantém em memória as informações mais acessadas recentemente de forma que acessos posteriores a esta mesma informação não irão precisar de acesso ao disco, bastando consultar os dados que já estão no buffer.
- **Gerenciamento do espaço em disco** - Responsável pelo gerenciamento dos recursos de armazenamento para a alocação física de dados. Ou seja, controla o sistema de arquivos localizando espaços livres ou liberados para armazenar os dados necessários.
- **Gerenciamento de Transação e Gerenciamento de Bloqueios** - O gerenciamento de transações é responsável por manter a consistência do Banco de Dados após a realização de uma transação. Uma transação é composta por um conjunto de operações que devem ser efetivadas no BD caso todas elas tenham sucesso, caso

contrário as operações realizadas até o momento de um erro ou uma falha devem ser desfeitas de maneira que a informação permaneça consistente.

O gerenciamento de bloqueios é responsável por controlar o acesso simultâneo aos dados. Em um SGBD várias transações estão ocorrendo ao mesmo tempo e muitas delas estão em estado concorrente aos dados do BD, ou seja, estão tentando obter ou modificar a mesma informação. Para que duas ou mais transações não modifiquem ao mesmo tempo o mesmo dado com informações diferentes, o SGBD bloqueia o registro sendo modificado liberando-o após a efetivação da operação.

Os componentes citados anteriormente utilizam intensamente as funções deste componente, pois necessitam garantir que o acesso e as modificações nos dados ocorram de maneira correta.

Ainda existem duas partes importantes:

- **A biblioteca de funções** - Armazena um conjunto de funções implementadas para dar apoio às operações do SGBD. Para isso foram desenvolvidas funções utilitárias (ex: funções de data e hora), expressões regulares entre outras. No PostgreSQL, a biblioteca é implementada na linguagem de programação C e as funções são carregadas dinamicamente de acordo com a necessidade de utilização. Por exemplo, no Sistema Operacional Microsoft Windows, essas bibliotecas seriam denominadas de *Dinamic Load Library* (DLL).
- **O Catálogo do Sistema** - Também chamado de meta-dados ou dicionário de dados, é responsável por armazenar informações sobre o próprio BD e seus objetos. Cada nova tabela, campo, usuário ou qualquer outro objeto criado no BD é registrado neste repositório. Da mesma maneira, caso esses objetos sejam removidos do BD a informação sobre eles também é removida desse repositório.

Para este trabalho, foi escolhido o PostgreSQL como SGBD de implementação dos aspectos temporais. Esse SGBD tem as características de orientação a objetos citadas na seção 4.1 tornando-o um SGBD objeto-relacional. Além disso, ele é de uso gratuito segundo os termos de licença *Berkeley Software Distribution* (BSD) e mais importante ainda é que possui o seu código-fonte aberto, o que possibilita a modificação ou extensão de suas funcionalidades internas.

A criação do novo tipo de dado *t_periodo* e a implementação dos aspectos temporais foram realizadas com os recursos do próprio SGBD.

4.3 O Módulo Temporal

Para a implementação dos aspectos temporais foi desenvolvido um módulo contendo o tipo de dado e o conjunto de funções e operações descritos na seção 3.2, que são necessários para auxiliar no gerenciamento das informações temporais.

O módulo temporal pode ser adaptado ao SGBD de forma a fazer parte da própria estrutura interna do SGBD. Diferente de algumas propostas, o módulo temporal não é uma API que faz a interface entre a aplicação e o SGBD. O módulo temporal faz parte dos processos e funções internas sendo requisitadas pelas aplicações por meio do SGBD.

Essa arquitetura tem a vantagem de tornar o módulo temporal transparente para a aplicação cliente. A aplicação pode utilizar os recursos implementados e contidos no módulo temporal de forma direta sem precisar efetuar nenhuma operação especial. Para utilizar a funcionalidade temporal oferecida pelo módulo, basta que a aplicação cliente conecte-se normalmente ao SGBD.

Esse módulo foi desenvolvido utilizando a linguagem de programação C e incluído na biblioteca do PostgreSQL. Assim como os demais módulos dentro dessa biblioteca, o módulo temporal será carregado dinamicamente quando o SGBD requisitar alguma de suas funções.

A figura 19 mostra a organização do módulo temporal, onde são utilizados alguns arquivos de cabeçalho (*header*) para o desenvolvimento e implementação das funções temporais deste trabalho.

Figura 19: Módulo Temporal

A figura 19 destaca apenas os principais arquivos necessários para a programação

do módulo temporal. Esses arquivos estão presentes na distribuição do próprio SGBD PostgreSQL.

Os detalhes de implementação do Módulo Temporal (`t_periodo.c`, `t_funcao.c`) estão descritos na seção 4.4.

4.4 Implementação

O suporte ao tempo válido implementado em um módulo como descrito na seção 4.3 pode ser adicionado a qualquer instalação pré-existente do PostgreSQL. Os programas, procedimentos (*procedures*), funções e gatilhos (*triggers*) continuam funcionando como antes sem necessitar de qualquer alteração em seu código, e têm a possibilidade de utilizar ou não esse novo conjunto de funcionalidades implementado e aplicado ao PostgreSQL.

4.4.1 Estrutura do *t_periodo*

O novo tipo de dado *t_periodo* pode ser implementado no PostgreSQL de duas formas. A primeira maneira utiliza os recursos da própria linguagem SQL implementados nesse SGBD. A segunda maneira utiliza uma linguagem de programação que possa fazer interface com o SGBD, por exemplo C, PHP, JAVA ou Perl.

- Utilização da Linguagem SQL

Utilizando a linguagem SQL é possível criar tipos de dados compostos. Esses tipos de dados podem ser especificados por uma lista de atributos e seus respectivos tipos de dados. Por exemplo, para criar o tipo de dado *t_periodo* deve ser utilizada a operação mostrada na figura 20.

```
CREATE TYPE t_periodo AS (  
    TVi datetime,  
    TVf datetime  
);
```

Figura 20: Criando um novo tipo de dado com SQL

Esse comando registra automaticamente no catálogo de sistema do SGBD (*System Catalog*) o tipo de dado criado. Com isso, ele se torna disponível para qualquer outra operação efetuada no BD em que foi criado. O exemplo da figura 21 mostra a criação da tabela funcionário especificada na figura 11. O atributo *t_validade* dessa tabela armazena informações sobre o tempo de vida do objeto funcionário. O

```
CREATE TABLE funcionario (  
    codigoFuncionario    INTEGER,  
    nomeFuncionario      VARCHAR(40),  
    emailFuncionario     VARCHAR(40),  
    t_validade           T_PERIODO  
);
```

Figura 21: Criando uma tabela com marca temporal

atributo foi criado utilizando o tipo de dado *t_perodo* definido com a linguagem SQL.

Apesar de ser mais rápida e fácil, a implementação utilizando a própria linguagem SQL do SGBD possui algumas limitações funcionais. Não é possível implementar as regras temporais de maneira que sejam diretamente vinculadas ao tipo de dado *t_perodo*. Para isso é necessário que cada classe criada com um atributo do tipo *t_perodo* também implemente as regras temporais. Ou seja, perde-se parcialmente a utilidade da estrutura apresentada na seção 3.2.

- Utilização de uma linguagem de programação

A segunda forma de estender a estrutura do PostgreSQL é desenvolver um módulo contendo o tipo de dado *t_perodo* e as operações necessárias. Para essa implementação foi utilizada a linguagem de programação C, pois ela permite que as regras temporais propostas para o tipo de dado *t_perodo* sejam implementadas juntamente com a estrutura de dados. A linguagem C é nativa da implementação do PostgreSQL, o que facilita muito a utilização de bibliotecas do próprio PostgreSQL e a comunicação com esse SGBD.

A implementação utiliza bibliotecas e funções disponíveis no PostgreSQL. Essas bibliotecas e funções permitem a integração do módulo temporal ao SGBD. O trecho de código da figura 22 mostra a implementação do tipo de dado *t_perodo* e as bibliotecas utilizadas.

Algumas bibliotecas utilizadas fazem parte da própria linguagem C padrão (*ctype.h*, *limits.h*, *float.h*, *time.h*), o restante são bibliotecas pertencentes a qualquer distribuição do SGBD PostgreSQL. Essas bibliotecas acompanham o SGBD justamente para permitir a implementação de novas funções e operações que visam estender a funcionalidade do PostgreSQL.

Na figura 22 observa-se que os elementos que compõem a estrutura "Periodo" são do tipo de dado *DateADT*. Esse tipo de dado faz parte das bibliotecas do PostgreSQL (*date.h*, *nabstime.h*) e está disponível para ser utilizado no desenvolvimento de outros programas. O tipo *DateADT* está preparado para processar e interpretar

```
#include "postgres.h"

#include <ctype.h>
#include <limits.h>
#include <float.h>
#include <time.h>

#include "access/hash.h"
#include "libpq/pqformat.h"
#include "miscadmin.h"
#include "parser/scansup.h"
#include "utils/builtins.h"
#include "utils/date.h"
#include "utils/nabstime.h"
#include "utils/timestamp.h"

typedef struct Periodo
{
    DateADT    tvl;
    DateADT    tvf;
} Periodo;
```

Figura 22: Código do *t_periodo*

valores de data e hora dos programas sendo utilizado nos tipos de dados *date*, *time* e *timestamp* já implementados e utilizados no PostgreSQL.

A criação de um novo tipo de dado base no PostgreSQL requer a implementação de funções para possibilitar as operações sobre esse tipo de dado.

Um tipo de dado definido pelo usuário (desenvolvedor), como é o caso do *t_periodo*, precisa sempre ter funções de entrada e saída vinculadas a ele. Essas funções determinam como serão mostradas as cadeias de caracteres (*strings*) para os usuários entrarem com dados ou visualizarem a saída de dados. Essas funções determinam também como o tipo de dado está organizado na memória da máquina.

A função de entrada recebe como argumento uma cadeia de caracteres terminadas por um caractere nulo e retorna para a memória da máquina a representação desse tipo possibilitando seu correto armazenamento. A função de entrada possibilita basicamente a manipulação dessa estrutura pelas operações de *insert* e *update* da linguagem SQL. No código da linguagem de programação, a função de entrada foi chamada de *periodo_in* (figura 23).

A função de saída faz o processo inverso da função de entrada, ela lê a estrutura armazenada na memória da máquina e transforma em uma cadeia de caracteres para

retornar para o usuário. Da mesma maneira que a função de entrada recebe um argumento passado como referência, a função de saída retorna a referência interna da cadeia de caracteres que representa o dado solicitado. A função de saída possibilita a recuperação da informação solicitada por meio da operação de seleção (*select*) da linguagem SQL.

```

1  Datum periodo_in(PG_FUNCTION_ARGS)
2  -{
3
4      char    *str = PG_GETARG_CSTRING(0);
5      char    tvi_ptr[MAXDATEFIELDS];
6      char    tvf_ptr[MAXDATEFIELDS];
7
8      Periodo  *result;
9
10     if (sscanf(str, " ( %s , %s )", tvi_ptr, tvf_ptr) != 2)
11     {
12         ereport(ERROR,
13             (errcode(ERRCODE_INVALID_TEXT_REPRESENTATION),
14              errmsg("Sintaxe inválida para periodo: \"%s\"", str)));
15     }
16
17     result = (Periodo *) palloc(sizeof(Periodo));
18     result->tvi = StringToDateADT(tvi_ptr);
19     result->tvf = StringToDateADT(tvf_ptr);
20
21     if (strcmp(tvi_ptr, "null") == 0) {
22         ereport(ERROR,
23             (errcode(ERRCODE_INVALID_TEXT_REPRESENTATION),
24              errmsg("TVi não pode ser nulo")));
25     }
26     else {
27         result->tvi = StringToDateADT(tvi_ptr);
28     }
29
30     if (result->tvi > result->tvf)
31     {
32         ereport(ERROR,
33             (errcode(ERRCODE_INVALID_TEXT_REPRESENTATION),
34              errmsg("Data inicial (TVi) deve ser menor ou igual a data final (TVf)")));
35     }
36
37     PG_RETURN_POINTER(result);
38 }

```

Figura 23: Código da função `periodo_in`

Com a necessidade de implementar as funções de entrada e saída para qualquer novo tipo de dado desenvolvido no PostgreSQL foi possível implementar também algumas das restrições temporais para o tipo de dado *t_perodo* descritas na seção 3.2. Na figura 23 é possível observar na linha 19 que a data inicial é comparada com *null* para garantir sempre que não seja atribuído valores nulos para essa data. E na linha 28, observa-se que os valores de TV_i e TV_f são verificados para garantir que TV_i não seja maior que TV_f . Caso ocorra uma violação dessas regras, o SGBD interrompe a operação e não efetiva o armazenamento do dado, mostrando para o usuário uma mensagem de erro.

A função de entrada `periodo_in` recebe como parâmetro uma cadeia de caracteres (*string*) do tipo "(TV_i , TV_f)". Por exemplo, "(10-jan-2007, 10-jul-2007)". Essa cadeia de caracteres é dividida em duas variáveis internas (*tvi_ptr* e *tvf_ptr*) que posteriormente são individualmente convertidas pela função `StringToDateADT` para o tipo de dado `DateADT` e armazenadas em uma variável do tipo *t_perodo*.

Com a estrutura do *t_periodo* e as respectivas funções de entrada e saída de dados já se apresenta um tipo de dado completo e utilizável pelo SGBD. O arquivo contendo o código deve ser compilado de forma a se transformar em uma biblioteca dinâmica que possa ser chamada e utilizada pelo SGBD. No caso do sistema operacional Windows essa biblioteca é denominada de *Dynamic Linked Library* (DLL) e no caso do Linux ela é denominada de *Shared Object* (SO).

Figura 24: Módulo Temporal Compilado

Para finalizar é necessário adicionar o tipo de dado *t_periodo* e as funções de entrada e saída ao catálogo de sistema. Para isso são utilizados comandos da linguagem SQL. A figura 25 apresenta o código SQL que cria no catálogo de sistema as funções de entrada (*periodo_in*), de saída (*periodo_out*) e o tipo de dado *t_periodo*.

Uma instância do PostgreSQL pode gerenciar ao mesmo tempo diversos Bancos de Dados (BD) que têm propósitos diferentes. Dessa forma, é importante destacar que a inclusão da estrutura do tipo de dado *t_periodo* e suas funções no catálogo de sistema do SGBD é exclusiva para um único Banco de Dados gerenciado por esse SGBD. Ou seja, para cada BD que deseja-se disponibilizar os recursos temporais, é necessário executar os

```
1 CREATE FUNCTION periodo_in(cstring)
2 RETURNS periodo AS
3 'periodo', 'periodo_in'
4 LANGUAGE 'c' IMMUTABLE STRICT;
5
6 CREATE FUNCTION periodo_out(periodo)
7 RETURNS cstring AS
8 'periodo', 'periodo_out'
9 LANGUAGE 'c' IMMUTABLE STRICT;
10
11 CREATE TYPE periodo (
12 INPUT=periodo_in,
13 OUTPUT=periodo_out,
14 );
```

Figura 25: Criação das funções de entrada, saída e o tipo de dado *t_perodo*

comandos da figura 25. Esse conceito também é válido para as implementações descritas nas seções seguintes.

Para que o tipo de dado *t_perodo* esteja sempre disponível para qualquer BD gerenciado pelo PostgreSQL, a implementação deveria ser um pouco diferente. Ao invés de utilizar um módulo que se adapta à estrutura do PostgreSQL, seria necessário modificar internamente o código do PostgreSQL e colocar o *t_perodo* junto com os outros tipos de dados básicos.

4.4.2 Implementação das Operações Temporais

O módulo temporal também contém a implementação das operações temporais descritas na seção 3.2.1 na figura 13. Essas operações auxiliam na manipulação dos dados armazenados em *t_perodo*.

Da mesma maneira que o tipo de dado *t_perodo*, as operações temporais podem ser implementadas utilizando a própria linguagem do SGBD, ou seja, a linguagem de manipulação de dados SQL. As operações também podem ser implementadas utilizando alguma linguagem de programação suportada diretamente pelo SGBD, como a linguagem C, JAVA, PHP, entre outras.

Novamente, a linguagem C foi escolhida, pois é mais flexível e permite a utilização de bibliotecas do próprio SGBD. Além disso, a estrutura de dados do *t_perodo* pode ser manipulada diretamente por essas funções.

As operações apresentadas na figura 13 são implementadas seguindo basicamente o mesmo padrão das operações exemplificadas na figura 26

A figura 26 mostra a implementação das operações *"equal"* (linhas de 1 a 12) e *"before"* (linhas de 14 a 25).

```

1  PG_FUNCTION_INFO_V1(periodo_equal);
2  Datum
3  periodo_equal(PG_FUNCTION_ARGS)
4  -{
5      Período    *a = (Período *) PG_GETARG_POINTER(0);
6      Período    *b = (Período *) PG_GETARG_POINTER(1);
7
8      if ((a->tvi == b->tvi) && (a->tvf == b->tvf))
9          PG_RETURN_BOOL(true);
10     else
11         PG_RETURN_BOOL(false);
12 }
13
14 PG_FUNCTION_INFO_V1(periodo_before);
15 Datum
16 periodo_before(PG_FUNCTION_ARGS)
17 -{
18     Período    *a = (Período *) PG_GETARG_POINTER(0);
19     Período    *b = (Período *) PG_GETARG_POINTER(1);
20
21     if (a->tvf < b->tvi)
22         PG_RETURN_BOOL(true);
23     else
24         PG_RETURN_BOOL(false);
25 }

```

Figura 26: Implementação das operações temporais

Basicamente, na implementação dessas funções temporais são declaradas duas variáveis do tipo $t_período$ (linhas 5 e 6, 18 e 19) que servirão para armazenar os intervalos de tempo a serem comparados. Em seguida, são realizadas as comparações necessárias em cada operação e retorna-se um valor booleano (verdadeiro ou falso); verdadeiro se as condições forem totalmente satisfeitas e falso caso o contrário.

Para exemplificar o funcionamento das operações temporais, apresenta-se abaixo duas cadeias de caracteres que representam dois intervalos de objetos distintos:

- Intervalo 1: "(25-jan-2007, 25-mai-2007)";
- Intervalo 2: "(25-jan-2007, 10-jun-2007)".

Considerando a função "equal" apresentada na figura 26, as duas cadeias de caracteres são utilizadas como parâmetros de entrada da função e interpretadas como sendo do tipo $t_período$, ou seja, os instantes de cada intervalo representam TV_i e TV_f respectivamente. O primeiro intervalo é associado ao ponteiro do tipo $t_período$ "a" (linha 5) e o segundo intervalo é associado ao ponteiro "b" (linha 6).

Assim, é possível comparar os intervalos como se mostra nas linhas seguintes da figura 26. Verifica-se se o TV_i do primeiro intervalo ($a->tvi$) representa uma data igual

ao TV_i do segundo intervalo ($b \rightarrow tvi$). Da mesma forma, o instante final do primeiro intervalo ($a \rightarrow tvf$) é comparado ao instante final do segundo intervalo ($b \rightarrow tvf$).

No exemplo, verifica-se que os dois intervalos não são iguais, pois o TV_i do primeiro e do segundo intervalo são iguais, mas o TV_f é diferente.

A função "before" assim como o restante das funções apresentadas na figura 13 tem a implementação semelhante a da função "equal".

Depois de ter criado e implementado essas funções no módulo temporal, é necessário que elas sejam adicionadas ao catálogo de sistema do SGBD para que se tornem disponíveis aos usuários. Da mesma forma que a estrutura de dados $t_periodo$ e suas funções de entrada e saída foram adicionadas ao catálogo de sistema, as funções temporais também são adicionadas por meio de instruções SQL executadas no BD alvo.

A figura 27 demonstra como cada função pode ser adicionada ao catálogo de sistema do SGBD.

```
1 CREATE FUNCTION periodo_equal(t_periodo, t_periodo)
2 RETURNS boolean
3 AS 'periodo', 'periodo_equal'
4 LANGUAGE C IMMUTABLE STRICT;
5
6 CREATE FUNCTION periodo_before(t_periodo, t_periodo)
7 RETURNS boolean
8 AS 'periodo', 'periodo_before'
9 LANGUAGE C IMMUTABLE STRICT;
```

Figura 27: Criação das funções temporais no SGBD

As funções podem receber qualquer nome desejado, desde que sejam seguidos os padrões de nomenclatura permitidos. Uma lista de parâmetros que essas funções aceitam receber deve ser declarado. No caso dessas funções temporais, essa lista é sempre composta de dois parâmetros do tipo $t_periodo$. Todas as funções temporais implementadas retornam um valor *booleano* (verdadeiro ou falso).

A próxima linha (linha 3 ou 8) da declaração indica o nome do módulo que contém a função a ser chamada e o nome da própria função implementada nesse módulo. E na linha seguinte é declarada a linguagem de programação utilizada para implementação do módulo que está sendo utilizada por essa função.

Após a criação das funções e a adição delas no catálogo de sistema é possível utilizá-las como qualquer outra função do SGBD. Por exemplo, para verificar se um intervalo temporal é anterior a outro utiliza-se a função "periodo_before" como descrito na figura 28.

A função da figura 28 compara o TV_f o primeiro parâmetro ("25-mai-2007") com o

```
1 SELECT periodo_before(' (25-jan-2007, 25-mai-2007)',
2                        '(26-mai-2007, 30-mai-2007)')
3 FROM dual;
```

Figura 28: Utilização da função `periodo_before`

TV_i do segundo parâmetro ("26-mai-2007"). Como a condição de comparação é satisfeita para essa função, o valor retornado é verdadeiro (*true*).

4.4.3 Implementação dos Operadores Temporais

A criação do tipo de dado *t_perodo* e a implementação das funções temporais vistas na seção 4.4.2 possibilitam adicionar ao contexto do SGBD os operadores temporais definidos por (ALLEN, 1983) e mostrados na figura 13.

Para criar novos operadores e adicioná-los ao catálogo de sistema do SGBD utiliza-se a instrução *CREATE OPERATOR* da linguagem SQL. Essa instrução não faz parte do padrão ANSI da linguagem e é implementada somente no PostgreSQL e em alguns outros SGBDs.

A figura 29 mostra um exemplo da criação de dois novos operadores no SGBD. A instrução SQL mostrada nessa figura, cria os operadores "equal" e "before" substituindo o nome por símbolos especiais. Respectivamente tem-se "==" e "<<". Essa substituição se deve ao fato de que a instrução não aceita nomes literais para novos operadores.

```
1 CREATE OPERATOR == (
2     leftarg    = t_perodo,
3     rightarg   = t_perodo,
4     procedure   = periodo_equal,
5     negator    = !=
6 );
7
8 CREATE OPERATOR << (
9     leftarg    = t_perodo,
10    rightarg   = t_perodo,
11    procedure   = periodo_before,
12    negator    = !<<
13 );
```

Figura 29: Criação dos operadores

Na criação dos operadores, é necessário indicar qual é o tipo de argumento que pode ser utilizado à esquerda do operador (linhas 2 e 9) e qual é o tipo de argumento que pode ser utilizado à direita do operador (linhas 3 e 10). No caso dos operadores temporais, ambos os argumentos devem ser do tipo *t_perodo*.

Também é necessário informar qual é a função que será utilizada na comparação desses dois argumentos (linhas 4 e 11). O primeiro operador criado representa a operação de "equal" logo a operação utilizada é "periodo_equal" mostrada na seção 4.4.2.

Opcionalmente é possível definir qual será o operador de negação de cada operador criado. Na figura 29, as linhas 5 e 12 mostram o operador de negação de cada um.

A tabela 9 mostra todos os operadores criados e seus respectivos símbolos, funções e operadores de negação.

Operador	Argumento	Símbolo	Negação
equal	t_perodo	==	! ==
before	t_perodo	<<	! <<
after	t_perodo	>>	! >>
meets	t_perodo		!
overlaps	t_perodo	<>	! <>
during	t_perodo	><	! ><
starts	t_perodo	>	! >
finishes	t_perodo	<	! <

Tabela 9: Operadores temporais

A criação desses operadores possibilita efetuar consultas nos objetos temporais do BD. A figura 30 mostra o exemplo de uma consulta utilizando a tabela criada na figura 21 e o operador "before".

```

1 SELECT *
2   FROM funcionario
3  WHERE t_validade << '(20-sep-2006, 20-dec-2006)'
```

Figura 30: Utilização do operador *before*

A consulta da figura 30 retorna todos os registros da tabela funcionário que satisfazem a condição de TV_f do atributo t_validade ser anterior ao valor '20-sep-2006'. Os outros operadores podem ser utilizados de forma idêntica.

Como descrito no final da seção 4.4.1, a adição de objetos no catálogo de sistema é exclusiva de apenas um BD alvo. Assim, também é necessário que os operadores descritos aqui sejam adicionados a cada BD que se deseja ter acesso ao conjunto de funções temporais.

4.4.4 Implementação das Restrições de Integridade Temporal

É importante que os valores dos objetos temporais permaneçam consistentes da mesma forma que os objetos não-temporais. Para isso, é necessária a implementação das restrições de integridade temporal descritas na seção 3.4.

A restrição sobre o intervalo temporal ($TV_i \leq TV_f$) foi implementada na própria criação do atributo temporal $t_periodo$ como mostrado na seção 4.4.1.

As restrições de chave primária e integridade de entidade foram desenvolvidas por meio da implementação de índices.

Os índices permitem organizar de maneira ordenada as informações armazenadas no BD. Para isso, é necessário efetuar a sobrecarga dos operadores relacionais ($<$, $\leq$, $=$, $\geq$, $>$) e utilizá-los para efetuar operações com o intervalo temporal $t_periodo$.

Todo o processo de implementar as funções na linguagem C, criar essas funções no catálogo de sistema do SGBD e associar operadores à essas funções é exatamente igual ao processo descrito nas seções 4.4.2 e 4.4.3.

A implementação de um índice no PostgreSQL requer uma etapa a mais, que é criar o índice no catálogo de sistema do SGBD por meio da instrução mostrada na figura 31

```

1 CREATE OPERATOR CLASS periodo_abs_ops
2   DEFAULT FOR TYPE t_periodo USING btree AS
3     OPERATOR      1      < (t_periodo, t_periodo),
4     OPERATOR      2      <= (t_periodo, t_periodo),
5     OPERATOR      3      = (t_periodo, t_periodo),
6     OPERATOR      4      >= (t_periodo, t_periodo),
7     OPERATOR      5      > (t_periodo, t_periodo),
8     FUNCTION      1      periodo_abs_cmp(t_periodo, t_periodo);

```

Figura 31: Criação do índice para ser utilizado na estrutura $t_periodo$

A implementação das funções utilizadas pelos operadores relacionais e a criação do índice no SGBD, define como será organizada e indexada as informações temporais que utilizarem um índice ou uma chave que seja composta por uma marca temporal.

As restrições de integridade referencial temporal devem ser implementadas de outra maneira. Poderia ser por meio da linguagem SQL, utilizando-se gatilhos (*triggers*) e procedimentos. Mas, dessa forma seria necessário que a cada objeto temporal criado, também fossem criadas funções para tratar essas restrições temporais.

Outra maneira, seria implementar essas restrições dentro do código do PostgreSQL, mas de uma maneira mais intrusiva. Ou seja, fora do módulo temporal.

A primeira solução não é ideal, pois o usuário continuaria tendo o trabalho de desenvolver gatilhos e procedimentos a cada novo objeto temporal criado, perdendo muito a transparência da solução apresentada pelo módulo temporal.

A implementação das restrições de integridade referencial fora do módulo temporal foge do propósito de criar um módulo à parte para implementar alguns aspectos temporais. Então foi deixado para as próximas etapas de implementação nos trabalhos seguintes.

5 AVALIAÇÃO DO MÓDULO TEMPORAL

O módulo temporal foi desenvolvido com o objetivo de facilitar e simplificar a implementação de sistemas que utilizam informações temporais.

Para avaliação do módulo temporal e verificação de suas funcionalidades é proposto um cenário experimental. Esse cenário é composto de parte de um sistema de controle da distribuição de petróleo pelo país.

5.1 Introdução

A cadeia de processos de produção de petróleo é extensa. De maneira geral, essa cadeia inicia-se na fase de prospecção de áreas com potencial petrolífero. Tem a fase de exploração dessas áreas, onde ocorre a perfuração do solo e montagem das estruturas para a produção de petróleo. A fase de produção, que inclui o processo de extração do petróleo a partir de sua origem. A fase de refino do petróleo é onde os derivados como gasolina, querosene e óleo diesel são produzidos. E por fim tem-se a fase de abastecimento, onde o petróleo e seus derivados são escoados para seus locais de consumo.

Este trabalho está considerando parte do sistema de abastecimento de petróleo, que é realizado através de dutos distribuídos por todo o país. Além dos dutos de óleo, o petróleo e seus derivados podem ser distribuídos através de caminhões, navios e trem. O meio mais utilizado realmente são os dutos de petróleo (figura 32), pois diminuem consideravelmente os custos da distribuição.

Um mesmo oleoduto pode transportar diferentes tipos de derivados de petróleo durante um período de tempo. Por exemplo, gasolina, óleo combustível e coque. Esses derivados são separados no duto pela própria composição química, mantendo um contato direto entre suas camadas. Quando os derivados são quimicamente muito semelhantes, há a necessidade de colocar um produto selador entre eles. A figura 33 mostra um exemplo dos derivados de petróleo sendo carregados dentro de um oleoduto.

A figura 33 mostra o transporte de três produtos derivados de petróleo (gasolina, óleo diesel e coque) e a água servindo de elemento selador entre a gasolina e o óleo diesel.

Figura 32: Oleodutos da Petrobras S/A

Para colocar os produtos dentro do duto é necessário um controle rigoroso em sua entrada. Não é permitido que dois produtos derivados sejam colocados ao mesmo tempo. Cada derivado é agendado para entrar no duto em intervalos de tempo diferentes. Assim, utilizando-se dos aspectos implementados no módulo temporal é possível controlar o fluxo de produtos ao longo do tempo dentro do oleoduto.

Figura 33: Derivados de petróleo dentro de um oleoduto

Além disso, a utilização de um SGBD Temporal permite que sejam registrados e armazenados os dados históricos sobre a utilização dos dutos. Essa informação histórica armazenada pode ser consultada para a tomada de decisão sobre o abastecimento de determinada região.

Por exemplo, uma determinada região rural consome mais óleo diesel no primeiro quadrimestre do ano, devido as colheitas feitas com grandes máquinas (tratores, caminhões, etc). Depois do período de colheita, o derivado de maior consumo torna-se a gasolina. Com os dados históricos armazenados é possível analisar a demanda de regiões e tomar ações para que os tipos de derivados sejam enviados pelos dutos segundo as necessidades da região nos diferentes períodos do ano. Também é possível avaliar e planejar o investimento em dutos, verificando se há a necessidade de construção de novos dutos.

5.2 Objetos do Sistema

Para o sistema de controle de dutos serão considerados apenas um subconjunto de seus objetos. Esses objetos representam exclusivamente os oleodutos do sistema de transporte de derivados de petróleo.

A figura 34 representa as classes que descrevem esse subconjunto de objetos. Nessa figura observa-se que a classe que representa os oleodutos do sistema tem o atributo validade, que indica o intervalo de tempo em que determinado oleoduto manteve seu funcionamento. Essa classe também é composta pela classe de agendamento, onde são registrados os vários derivados que passam pelo duto, a quantidade desses derivados e o período em que deve ocorrer a operação de entrada do produto no duto.

Figura 34: Classes do sistema de dutos

A criação das tabelas no SGBD segue a descrição dos comandos apresentados na figura 35. Utilizando o tipo de dado *t_periodo* apresentado no capítulo 3, é possível criar os objetos com marcas temporais que representam o tempo de validade de cada instância do objeto. As linhas 5 e 14 mostram os atributos que representam essas marcas temporais.

```

1 CREATE TABLE oleoduto (
2     id          INT,
3     nome        VARCHAR(50),
4     localizacao VARCHAR(50),
5     validade    T_PERIODO,
6     CONSTRAINT pk_oleoduto
7         PRIMARY KEY (id, validade)
8 );
9
10 CREATE TABLE agendamento (
11     id_duto     INT,
12     derivado    VARCHAR(50),
13     quantidade  DECIMAL(10,2),
14     validade    T_PERIODO,
15     CONSTRAINT pk_agendamento
16         PRIMARY KEY (id_duto, validade),
17     CONSTRAINT fk_agendamento_oleoduto
18         FOREIGN KEY (id_duto, validade)
19         REFERENCES oleoduto
20 );
  
```

Figura 35: Tabelas do sistema

A chave primária da tabela "oleoduto" identifica uma única instância dessa classe

(cada objeto). Essa chave primária é composta pelos atributos "id" e "validade". Para garantir que os objetos sejam únicos no sistema, a tabela "oleoduto" precisaria ter apenas o atributo "id" como chave primária, mas foi necessária a inclusão do atributo "validade" nessa chave para que pudesse ser mantida a integridade de entidade em relação aos valores das marcas temporais do atributo composto ("agendamento") e do objeto "oleoduto", onde o primeiro deve estar contido no período do segundo. Isso é devido ao tipo de implementação que foi realizada no módulo temporal do SGBD, é não é determinada diretamente pela definição da chave primária.

A tabela "agendamento" representa um atributo temporal que é mapeado para uma entidade fraca no modelo ER. Por isso, sua chave primária precisou ser composta pelo atributo "id_duto" da tabela "oleoduto" e mais o tempo válido para essa classe. Essa chave indica que não pode ocorrer a entrada de diferentes derivados de petróleo em um mesmo duto no mesmo período. Se por acaso precisar que sejam misturados dois ou mais derivados (gasolina + álcool), essa mistura não é realizada diretamente nos dutos, elas são misturadas em tanques específicos para isso.

A chave estrangeira da tabela "agendamento" garante a integridade de entidade temporal (seção 3.5.1) entre as tabelas. Isso significa que só poderão existir agendamentos em que os valores das marcas temporais representadas por `agendamento.validade` estejam dentro (sejam um subconjunto) dos valores do atributo `validade` da tabela "oleoduto" (`oleoduto.validade`).

É necessário notar aqui que o relacionamento entre essas duas classes é um relacionamento de composição, onde o objeto `agendamento` representa um atributo temporal do objeto `oleoduto`. Por isso, ao invés de aplicar a restrição de integridade referencial a restrição de integridade de entidade é aplicada aqui.

5.3 Manipulação dos Dados Temporais

Utilizando as tabelas criadas com as operações mostradas na figura 35 é possível efetuar as operações para registrar novos valores para os dois objetos.

Para manipular essas informações foi utilizada uma interface (GUI) denominada `pgAdminIII`. Essa interface é específica para auxiliar no gerenciamento do PostgreSQL e permite efetuar todas as operações SQL nesse SGBD. O `pgAdminIII` também é de uso gratuito e pode ser encontrado na internet.

As figuras 36 e 37 mostram a inclusão de novos registros por meio da operação *INSERT* da linguagem SQL.

As operações de *insert* da figura 36 incluem três novos registros na tabela "oleo-

Figura 36: Inclusão de novos registros na tabela oleoduto

duto". Em seguida é efetuada uma operação de consulta (*select*), que mostra os dados já registrados no SGBD.

É necessário observar na figura 36 os valores do atributo validade, que se referem ao tempo em que cada registro é valido no mundo real (tempo de validade). Esses valores são importantes para constituir as restrições de integridade de entidade e as restrições de integridade referencial temporal.

As operações de *insert* da figura 37 incluem quatro novos registros na tabela "agendamento". As três primeiras operações incluem agendamentos para o oleoduto da localidade de Paulínia, a última operação de *insert* inclui um agendamento para o oleoduto da localidade de Cubatão. Em seguida é efetuada uma operação de consulta (*select*) na tabela "oleoduto" fazendo junção com a tabela "agendamento". O resultado mostra os oleodutos existentes e seus respectivos agendamentos.

O atributo "validade" da tabela "oleoduto" faz parte da chave primária juntamente com o atributo "id_duto". Isso permite garantir que cada oleoduto tenha apenas um agendamento em um determinado período. Por exemplo, na figura 37 os agendamentos feitos para o duto da localidade de Paulínia não se sobrepõem em momento algum. Se por acaso for necessária a modificação de um agendamento, antecipando ou postergando a entrada de um derivado no duto, é necessário efetuar uma operação de atualização. Essa operação também estará susceptível às restrições impostas pelo tipo de dados *t_perodo*.

Esse atributo "validade" também faz parte da chave estrangeira da tabela "agendamento" que referencia a tabela "oleoduto". Isso garante que cada agendamento registrado tenha seu período de validade dentro do período de validade estipulado para o respectivo oleoduto. Por exemplo, os valores do atributo validade para os agendamentos feitos para

Figura 37: Inclusão de novos registros na tabela agendamento

o oleoduto de Paulínia devem estar entre 10 de março de 2000 e 31 de dezembro de 2007. Dessa maneira, é mantida a integridade existencial temporal.

Os operadores implementados permitem efetuar operações de seleção (SELECT) baseados nos valores das marcas temporais. Por exemplo, a figura 38 obtém todos os objetos (registros) que contenham a marca temporal dentro do período (2000-03-09, 2000-03-14).

Figura 38: Operação de seleção com o operador *during*

5.4 Violação das Restrições de Integridade Temporal

A manipulação das informações temporais no SGBD podem violar as regras de integridade temporal descritas na seção 3.3.

A chave primária temporal definida na tabela "oleoduto" é composta pelos atributos "id" e "validade". Qualquer operação que insira valores repetidos para essa chave deve ser rejeitada pelo SGBD. No caso da tabela "oleoduto", um valor repetido seria a inserção de um registro com um valor para o atributo "validade" que sobreponha qualquer outro período já existente para um mesmo valor de "id".

Figura 39: Violação da restrição de chave primária (unicidade)

A figura 39 mostra uma operação violando a restrição de chave primária temporal e sendo rejeitada pelo SGBD. A operação de inserção é rejeitada, pois o valor do atributo "validade" que está sendo inserido sobrepõe um valor já existente para o mesmo objeto no SGBD. A implementação dessa restrição contempla algumas classes de aplicações temporais, onde os intervalos são disjuntos. Algumas outras classes de aplicações temporais permitem que valores das marcas temporais se sobreponham.

Considerando também que o atributo validade da tabela agendamento faz parte da chave primária dessa tabela, é necessário que seus valores não sejam nulos. A operação da figura 40 é rejeitada, pois o tempo de validade inicial é nulo (*null*).

Figura 40: Violação da chave primária (nulo)

A chave estrangeira definida na tabela "agendamento" (figura 35) é composta pelos atributos "id_duto" e "validade". Ela mantém a integridade existencial temporal com a tabela "oleoduto". Assim, qualquer conjunto de valores inseridos para um registro na tabela "agendamento" deve possuir os valores de sua chave estrangeira existentes na tabela referenciada, no caso a tabela "oleoduto".

As figuras 41, 42 e 43, mostram alguns exemplos de violação da integridade existencial temporal. As operações mostradas nessas figuras tentam inserir registros de agendamento para o oleoduto da localidade de Mauá.

Figura 41: Violação da restrição de integridade existencial temporal (*before*)

Como mostrado na figura 36, o oleoduto de Mauá possui o valor do tempo de validade igual a "(10-fev-2010, 30-nov-2025)", ou seja, é um duto que ainda não entrou em operação.

A operação da figura 41 tenta inserir um agendamento com o valor do tempo de validade anterior (*before*) ao tempo de validade do objeto oleoduto de Mauá.

Figura 42: Violação da restrição de integridade existencial temporal (*after*)

A operação da figura 42 tenta inserir um agendamento com um valor do tempo de validade posterior (*after*) ao tempo de validade do objeto oleoduto de Mauá.

A operação da figura 43 tenta inserir um agendamento com um valor do tempo de validade que inicia-se dentro (*overlaps*) do tempo de validade do objeto oleoduto de Mauá, mas termina depois do final de seu período.

Figura 43: Violação da restrição de integridade existencial temporal (*overlaps*)

Além das inserções, qualquer operação de atualização (UPDATE) que modifique as marcas temporais do objeto oleoduto é submetida ao conjunto de restrições do módulo temporal. Assim, caso sejam modificadas as marcas temporais de um objeto de forma que os valores fiquem inconsistentes com os agendamentos já cadastrados, então essa operação deve ser rejeitada.

Uma outra violação de integridade que pode ocorrer é apresentada na figura 44. Essa figura é rejeitada pelo SGBD, pois infringe a regra $TV_i \leq TV_f$ definida sobre o intervalo temporal.

Figura 44: Violação da restrição de domínio

No caso do exemplo da figura 44, o tempo de validade inicial ('05-fev-2007') é maior que o tempo de validade final ('10-mar-2005').

A operação de remoção (*delete*) pode ocasionar uma violação na restrição de integridade referencial temporal, quando dois objetos possuem um relacionamento entre si e o objeto referenciado é removido.

Essa operação de remoção também pode ocasionar uma violação de integridade existencial temporal, quando um atributo temporal depende de um objeto temporal e ele é removido. Por exemplo, caso um objeto oleoduto fosse removido e existissem agendamentos relacionados a ele.

As operações de atualização (*update*) podem infringir todas as restrições mostradas para as operações de inserção e remoção. Assim, os exemplos anteriores se aplicam a essas operações.

6 CONCLUSÃO E TRABALHOS FUTUROS

A proposta apresentada nesse trabalho seguiu uma abordagem delimitada entre os escopos do conhecimento nas áreas de Sistemas e Bancos de Dados Temporais. Para esse fim, o objeto do trabalho foi um módulo temporal que implementa alguns dos aspectos temporais propostos na literatura.

6.1 Sobre o Módulo Temporal

A implementação dos aspectos temporais em um módulo que se encaixa no conjunto de funcionalidades do SGBD mostrou a importância de manter a semântica temporal inteiramente pertencente e gerenciada pelo próprio SGBD. Pois, dessa maneira, torna a complexidade temporal transparente para o usuário e facilita a utilização dos recursos desse módulo.

A maneira como o módulo temporal foi projetado e implementado permite que novas funcionalidades temporais sejam acrescentadas ao contexto do SGBD. Por exemplo, funções de agregação temporal (max, min, avg) podem ser adicionadas no módulo da mesma forma e facilidade que os outros aspectos foram implementados.

A implementação das características temporais em um módulo, que foi desenvolvido em uma camada separada da arquitetura interna do SGBD, diminui ou elimina a possibilidade que os erros de programação do módulo temporal interfiram negativamente em outras aplicações que usam o mesmo SGBD, mas que não utilizam os recursos temporais.

O novo tipo de dado *t_periódico*, implementado nesse módulo, ajuda a gerenciar as marcas temporais de uma forma mais transparente para o usuário e facilita o desenvolvimento de aplicações temporais que utilizem esse SGBD. Assim, a representação dos valores temporais tornou-se melhor, pois agrupou em um único atributo os limites do intervalo temporal. Além disso, foi possível implementar algumas restrições pertinentes ao contexto temporal descritas na seção 3.3 e intrínsecas ao tipo de dado *t_periódico*.

Por outro lado, a desvantagem desse tipo de implementação é que o módulo temporal fica completamente dependente da arquitetura e da estrutura interna do SGBD (PostgreSQL), o que acaba dificultando a portabilidade entre diferentes plataformas.

6.2 Sobre o SGBDOR

As características de um Sistema Gerenciador de Banco de Dados Objeto Relacional (SGBDOR) apresentadas no capítulo 3 salientaram a possibilidade de estender e modificar as funcionalidades do SGBD.

Este trabalho utilizou essas características para implementar alguns dos mais importantes aspectos temporais. Assim, foi demonstrada a importância das funcionalidades de um SGBD poderem ser alteradas, pois com isso, pode absorver as demandas emergentes.

Apesar do SGBDOR permitir a aninhamento de tabelas, não foi possível representar os atributos temporais por meio dessa característica. Isso se deve ao fato de que a tabela aninhada não suporta múltiplos registros, ela suporta somente um único registro composto por todos os atributos dessa tabela aninhada.

Por causa disso, a implementação das classes apresentadas no exemplo do capítulo 5 teve que ser mapeada para o modelo relacional e o atributo temporal agendamento convertido em uma tabela que mantém um relacionamento com a tabela pai (tabela oleoduto).

Esse fato acaba não sendo muito transparente ao usuário (desenvolvedor) que deve se preocupar em criar uma tabela para cada atributo temporal e fazer os devidos relacionamentos com a tabela pai.

Isso também acarreta a necessidade de serem criadas chaves primárias constituídas pela marca temporal nas tabelas que são referenciadas, pois as chaves estrangeiras das tabelas que representam os atributos temporais só podem referenciar um conjunto de atributos que seja único.

Para viabilizar o desenvolvimento desse trabalho foi escolhido um SGBDOR livre de licença e com o código-fonte aberto (*open source*). Então, com a utilização do PostgreSQL a parte de implementação foi alcançada no capítulo 4 e mesmo com as limitações desse SGBD, seus recursos foram suficientes para poder cumprir com os objetivos deste trabalho.

6.3 Sobre os Bancos de Dados Temporais

As pesquisas na área de sistemas temporais continuam a evoluir, pois ainda há uma forte necessidade dos SGBDs serem capazes de gerenciar adequadamente os aspectos temporais.

Uma outra forte pesquisa em Bancos de Dados é o gerenciamento e armazenamento de informações geoespaciais. Atualmente, os BDT estão sendo estudados juntamente com esses Bancos de Dados Geoespaciais (BDG) (KHATRI; RAM; SNODGRASS, 2004). Pois, dessa maneira, oferecem recursos para armazenar imagens geoespaciais vinculadas às marcas

temporais.

Para a linguagem de consulta (SQL) já existe uma especificação completa (SNODGRASS et al., 1994) dos comandos temporais. Muitos deles já estão incorporados no padrão SQL.

6.4 Trabalhos Futuros

Considerando o aspecto histórico dos Bancos de Dados Temporais e o armazenamento de um volume maior de dados é necessário investigar algoritmos de busca de informações mais eficientes. A própria estrutura do tipo de dado *t_perodo* necessita de algoritmos próprios para operações em intervalos de tempo.

Como parte desses algoritmos de busca de informações, a indexação dos dados temporais deve ser pesquisada para verificar as possibilidades de melhoria na utilização de índices sobre períodos. Assim como os outros tipos de dados existentes no SGBD, o intervalo temporal *t_perodo* deveria ter a possibilidade de utilizar diferentes métodos de acesso, determinados pelas operações e pelos valores utilizados.

O módulo temporal foi desenvolvido de maneira não-intrusiva, ou seja, o código-fonte do núcleo do SGBD permaneceu inalterado. Todas as implementações utilizaram-se das bibliotecas disponíveis para esse SGBD, mas não precisaram alterar nenhum dos arquivos originais do PostgreSQL.

Contudo, para efetuar a implementação completa de um SGBD temporal, deve ser utilizada uma maneira mais intrusiva de implementação. Ou seja, alterações no núcleo original do PostgreSQL devem ser realizadas.

A principal alteração que deve ser feita de forma intrusiva é na parte de interpretação dos comandos SQL. Essas alterações teriam o objetivo de implementar as operações e cláusulas determinadas pela TSQL2 (SNODGRASS et al., 1996a, 1996b). Entre essas alterações está a implementação das operações de *insert*, *update* e *delete*, que permitiriam trabalhar mais facilmente com o intervalo temporal. Além disso, essas operações devem ser modificadas para que seja preservado o histórico das informações, assim como descrito na seção 2.2.2.

Também deve ser alterado a questão da tabela aninhada para que seja possível representar o atributo temporal mais facilmente e de forma realmente transparente ao usuário.

Com a finalização dessas implementações, pretende-se submeter o módulo temporal para as contribuições do PostgreSQL. Assim, o módulo ficará disponível para a comunidade de usuários desse SGBD. Além da disponibilidade de utilização, os desenvolvedores que participam do aperfeiçoamento do PostgreSQL podem auxiliar no crescimento do módulo temporal.

REFERÊNCIAS BIBLIOGRÁFICAS

- ALLEN, J. F. Maintaining knowledge about temporal intervals. *Communications of the ACM*, v. 26, n. 11, p. 832–843, 1983. ISSN 0001-0782.
- ASTRAHAN, M. M.; BLASGEN, M. W.; CHAMBERLIN, D. D.; ESWARAN, K. P.; GRAY, J. N.; GRIFFITHS, P. P.; KING, W. F.; LORIE, R. A.; MCJONES, P. R.; MEHL, J. W.; PUTZOLU, G. R.; TRAIGER, I. L.; WADE, B. W.; WATSON, V. System r: Relational approach to database management. *ACM Transactions on Database Systems*, v. 1, n. 2, p. 97–137, 1976. ISSN 0362-5915.
- ASTRAHAN, M. M.; CHAMBERLIN, D. D. Implementation of a structured english query language. *Communications of the ACM*, v. 18, n. 10, p. 580–588, 1975. ISSN 0001-0782.
- BOHLEN, M. *Valid Time Integrity Constraints*. Tucson, AZ, 1994.
- CHEN, C. X.; KONG, J.; ZANIOLO, C. Design and implementation of a temporal extension of sql. *icde*, IEEE Computer Society, Los Alamitos, CA, USA, v. 00, p. 689, 2003. ISSN 1063-6382.
- CHEN, P. P. The entity-relationship model - toward a unified view of data. *ACM Transaction on Database Systems.*, v. 1, n. 1, p. 9–36, 1976.
- CHENGALUR-SMITH, I. N.; BALLOU, D. P.; PAZER, H. L. The impact of data quality information on decision making: An exploratory analysis. *IEEE Trans. Knowl. Data Eng.*, v. 11, n. 6, p. 853–864, 1999.
- CODD, E. F. A relational model of data for large shared data banks. *Communication of the ACM*, p. 377–387, 1970.
- CODD, E. F. *The Relational Model for Database Management, Version 2*. Massachusetts: Addison-Wesley, 1990. 567 p. ISBN 0-201-14192-2.
- CORDEIRO, R. L. F.; SANTOS, C. S. dos; EDELWEISS, N.; GALANTE, R. de M. Classificação de restrições de integridade em bancos de dados temporais de versões. 2005.
- DATE, C. J. *Introdução a Sistemas de Bancos de Dados*. São Paulo: Editora Campus, 2004. 900 p. ISBN 85-352-1273-6.
- DATE, C. J.; DARWEN, H.; LORENTZOS, N. *Temporal Data and the Relational Model*. San Francisco: Morgan Kaufmann, 2002. 422 p. ISBN 1-55860-855-9.
- EDELWEISS, N. *Banco de Dados Temporais: Teoria e Prática*. [S.l.], 1998. Disponível em: <<http://www.inf.ufrgs.br/nina/ArtigosNina/JAI1998.pdf>>.
- EDELWEISS, N.; HUBLER, P.; MORO, M.; DEMARTINI, G. A temporal database management system implemented on top of a conventional database. *International Conference of the Chilean Computer Science Society*, v. 00, p. 58, 2000. ISSN 1522-4902.

- GERTZ, M.; LIPECK, U. W. Temporal integrity constraints in temporal databases. In: *Proceedings of the International Workshop on Temporal Databases*. London, UK: Springer-Verlag, 1995. p. 77–92. ISBN 3-540-19945-4.
- GORALWALLA, A.; OZSU, M. T.; TANSEL, A. U. Experimenting with temporal relational databases I. In: *Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM)*. Baltimore, MD: [s.n.], 1995. p. 296–303. Disponível em: <citeseer.ist.psu.edu/goralwalla95experimenting.html>.
- GORALWALLA, I. A.; TANSEL, A.; ÖZSU, M. T. *An Empirical Study of the Performance of Temporal Relational Databases*. [S.l.], 1994. Disponível em: <[ftp://menaik.cs.ualberta.ca/pub/TechReports/1994/TR94-15/TR94-15.ps.Z](http://menaik.cs.ualberta.ca/pub/TechReports/1994/TR94-15/TR94-15.ps.Z)>.
- GREGERSEN, H.; JENSEN, C. S. Temporal entity-relationship models - a survey. *IEEE Transactions on Knowledge and Data Engineering*, v. 11, n. 3, p. 464–497, 1999.
- ISO/IEC. *ISO/IEC 9075 - Information Technology - Database Languages - SQL*. [S.l.], 1999.
- JENSEN, C. S.; CLIFFORD, J.; GADIA, S. K.; SEGEV, A.; SNODGRASS, R. T. A glossary of temporal database concepts. *SIGMOD Record*, ACM, v. 21, n. 3, p. 35–43, sep 1992.
- JENSEN, C. S.; SNODGRASS, R. T. Temporal data management. *IEEE Transactions on Knowledge and Data Engineering*, v. 11, n. 1, p. 36–44, 1999.
- JENSEN, C. S.; SNODGRASS, R. T.; TANSEL, A. U. *Temporal Databases: Theory, Design, and Implementation*. Redwood City: Benjamin/Cummings, 1994. 633 p. ISBN 0-8053-2413-5.
- KHATRI, V.; RAM, S.; SNODGRASS, R. T. Augmenting a conceptual model with geospatiotemporal annotations. *IEEE Transactions on Knowledge and Data Engineering*, v. 16, n. 11, p. 1324–1338, 2004. ISSN 1041-4347.
- KLASSEN, C. What is a temporal database. 1999. Disponível em: <<http://citm.utdallas.edu/publications/whitepapers/wptemporalbb.htm>>.
- KLIN, N. An update of the temporal database bibliography. *sigmod*, v. 22, n. 4, p. 66–80, dec 1993.
- KUNG, C. H. On verification on database temporal constraint. 1985.
- NJOVU, C.; IBRAHIM, M. T. A design pattern approach to bitemporal data modelling. IEEE Computer Society, Washington, DC, USA, p. 758, 2003.
- OLIVEIRA, A. M. de. *Funcionalidades Temporais do Banco de Dados Oracle e Mecanismos para Consultas de Tempo de Validade*. Rio Grande do Sul: [s.n.], 2003.
- ORACLE. *An Oracle White Paper - Oracle9i Flashback Query*. [S.l.], 2002. Disponível em: <<http://www.oracle.com/technology/deploy/availability/pdf/FlashbackQueryBWP.pdf>>.
- OZSOYOGLU, G.; SNODGRASS, R. T. Temporal and real-time databases: A survey. *IEEE Transactions on Knowledge and Data Engineering*, v. 7, n. 4, p. 513–532, 1995.

- PINHEIRO, S. F. *PTV - Pacote de Tempo de Validade para o SGBD Oracle*. Rio Grande do Sul: [s.n.], 2003.
- SANTOS, C. da S. *Implementação de Bancos de Dados Temporais com Versionamento de Esquemas: um estudo comparativo*. Rio Grande do Sul: [s.n.], 2003.
- SANTOS, G. F. *Realização de Consultas Visuais a Banco de Dados Temporais*. Campina Grande - PB: [s.n.], 2000.
- SILBERSCHATZ, A.; KORTH, H. F.; SUDARSH. *Sistema De Banco De Dados*. São Paulo: Makron Books, 1999. 806 p. ISBN 8-53461-073-8.
- SNODGRASS, R. T. *Developing Time-Oriented Database Applications in SQL*. San Francisco: Morgan Kaufmann, 1999. 504 p. ISBN 1-55860-436-7.
- SNODGRASS, R. T.; AHN, I.; ARIAV, G.; BATORY, D. S.; CLIFFORD, J.; DYRESON, C. E.; ELMASRI, R.; GRANDI, F.; JENSEN, C. S.; KAHER, W.; KLINE, N.; KULKARNI, K. G.; LEUNG, T. Y. C.; LORENTZOS, N. A.; RODDICK, J. F.; SEGEV, A.; SOO, M. D.; SRIPADA, S. M. Tsql2 language specification. *SIGMOD Record*, v. 23, n. 1, p. 65–86, 1994.
- SNODGRASS, R. T.; BÖHLEN, M. H.; JENSEN, C. S.; STEINER, A. *Adding Transaction Time to SQL/Temporal*. [S.l.], 1996. Disponível em: <<http://www.cs.aau.dk/~csj/Thesis/pdf/chapter27.pdf>>.
- SNODGRASS, R. T.; BÖHLEN, M. H.; JENSEN, C. S.; STEINER, A. *Adding Valid Time to SQL/Temporal*. [S.l.], 1996. Disponível em: <<http://www.cs.aau.dk/~csj/Thesis/pdf/chapter26.pdf>>.
- SNODGRASS, R. T.; KUCERA, H. Rationale for a temporal extension to sql. In: *The TSQL2 Temporal Query Language*. [S.l.: s.n.], 1995. p. 3–18.
- SOUZA, S. N. A. *Especificação de um Modelo de Dados Bitemporal Orientado a Objetos*. Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo (EPUSP), São Paulo, 1998.
- STEINER, A. *A Generalisation Approach to Temporal Data Models and their Implementations*. Tese (Doutorado) — Swiss Federal Institute of Technology, Zurich, 1998.
- STEINER, A. *TimeDB 2.0*. [S.l.], 1999. Disponível em: <<http://www.cs.arizona.edu/projects/timecenter/>>.
- STUMPF, S.; MCDONNELL, J. Data, information and knowledge quality in retail security decision making. p. 2–4, 2003.
- TANSEL, A. U. Integrity constraints in temporal relational databases. p. 460–464, 2004.
- TANSEL, A. U. Temporal data modeling and integrity constraints in relational databases. In: *ISCIS*. [S.l.: s.n.], 2004. p. 459–469.
- WAND, Y.; WANG, R. Y. Anchoring data quality dimensions in ontological foundations. *Communications of the ACM*, ACM Press, New York, NY, USA, v. 39, n. 11, p. 86–95, 1996.

WEISS, R. *How Oracle Database 10G Revolutionizes Availability and Enables The Grid*. [S.l.], 2005. Disponível em: <<http://www.oracle.com/technology/tech/grid/collateral/10gAvailability.pdf>>.