Incorporação da Dimensão Temporal a Bancos de Dados Orientados a Objetos

Lincoln César Medina de Oliveira
Claudia Bauzer Medeiros (Orientadora)
Departamento de Ciência da Computação
IMECC - UNICAMP

Incorporação da Dimensão Temporal a Bancos de Dados Orientados a Objetos

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Lincoln César Medina de Oliveira e aprovada pela Comissão Julgadora.

Campinas, 23 de julho de 1993

Prof. Dr. 

Claudia Maria Bauzer Medeiros

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

Aos meus pais, que me apoiaram em mais esta etapa de minha vida, e tornaram possível a realização deste trabalho.

Agradecimentos

Este trabalho não pode, de modo algum, ser considerado uma realização solitária. Aproveito este espaço para agradecer a todos aqueles que me auxiliaram na sua execução. Em particular, gostaria de agradecer:

- à minha orientadora Claudia, por suas sugestões e disposição permanente em me atender;
- ao meu amigo Cléber, meu companheiro mais constante durante todo este período;
- a toda turma do vôlei, dos pioneiros aos mais recentes, pelos momentos de descontração que me ajudaram a enfrentar a rotina semanal;
- aos companheiros das salas de estudo, em especial a Adauto, Heitor, Mário e Thierson;
- a todos aqueles com quem convivi mais estreitamente, e com os quais partilhei os instantes mais agradáveis deste mestrado (Cecília, Ivone, Paulo, Inesinha, Carrilho, Atta, Nabor ...).

Gostaria ainda de agradecer ao Conselho Nacional de Desenvolvimento Científico e Tecnológico — CNPq — pelo suporte financeiro à execução do programa de Mestrado.

Sumário

Bancos de dados temporais têm sido alvo de intensa pesquisa nas últimas duas décadas. Embora vários resultados já tenham sido obtidos para sistemas relacionais temporais, existem poucas propostas que considerem a incorporação de facilidades temporais a bancos de dados orientados a objetos. Este trabalho visa contribuir para o desenvolvimento desta área. Para tanto, a dissertação apresenta os seguintes resultados originais: uma ampla revisão bibliográfica sobre modelos e linguagens temporais; proposta de um novo modelo temporal (TOODM), baseado no paradigma de orientação a objetos; e especificação de uma linguagem de consulta (TOOL) para o modelo proposto.

Abstract

The last two decades have witnessed an intensive research on temporal databases. Although several results have already been achieved for temporal relational systems, there are few proposals considering the incorporation of the temporal dimension into object oriented databases. The present work contributes to the development of this area. This dissertation presents the following original results: a broad survey on temporal models and languages; the proposal of a new temporal model (TOODM), based on the object oriented paradigm; and the specification of a query language (TOOL) for the proposed model.

Conteúdo

Capítulo 1

Introdução

Um dos objetivos básicos de um banco de dados (BD) é modelar o mundo real da melhor forma possível. Esta necessidade tem impulsionado a pesquisa de modelos de BD's, ocasionando assim o surgimento de vários modelos de dados, e.g., modelos em rede, relacional, entidade-relacionamento (ER) e orientado a objetos (OO). Todavia estes modelos têm historicamente negligenciado um aspecto fundamental na modelagem do mundo real: o tempo. O gerenciamento de informações ligadas ao tempo assim como a manutenção de dados passados de BD's são essenciais para um grande número de aplicações, tais como, automação de escritórios, sistemas de suporte a decisão, aplicações CAD, CAM e CIM, processamento de dados científicos, estatísticos e cartográficos, reservas aéreas, aplicações médicas e contábeis, entre outras. Em virtude disto, o tratamento do aspecto tempo tem sido efetuado de maneira *ad hoc*, com a manutenção de *backups* e *logs* de correção e a manipulação de informações temporais sendo implantada no código das próprias aplicações.

A necessidade de um tratamento mais sistêmico da questão do tempo, aliada ao crescimento da capacidade e da queda dos custos dos meios de armazenamento de dados, provocou o surgimento de um novo campo de pesquisas na área de BD's: os *bancos de dados temporais* (BDT's).

1.1 Motivação e Objetivos da Dissertação

A necessidade de se prover suporte a dados temporais no nível de sistema gerenciador de banco de dados (SGBD) foi percebida há mais de uma década pela comunidade científica. Desde então o tema vem recebendo atenção crescente, dando origem a um grande número de trabalhos publicados na área. Tais trabalhos enfocam questões como: o estudo da semântica do tempo, a extensão de modelos tradicionais, a proposta de novos modelos, a extensão da álgebra e do cálculo relacional, a proposta de linguagens de consulta e a implementação de modelos propostos, sempre visando BDT's.

Snodgrass relaciona em [Sno86] dezenas de grupos de pesquisa que estão ou estiveram em contato com o tema; em [Sno90] relata a existência de mais de 350 artigos relacionando processamento de informações e tempo. Devido à grande quantidade de publicações sobre o tema, surgiram algumas bibliografias sobre o assunto (e.g., [McK86, Soo91]) e comparações entre modelos temporais (e.g., [CC87, TC90, MP91, MS91, Ari86, Gad88, GY88, SS88, Sno87, TAO90]). Grande parte destas análises comparativas está inclusa em artigos de descrição de modelos temporais, sendo normalmente bastante restritas em relação ao número de modelos e critérios analisados (e.g., [Ari86, Gad88, GY88, SS88, Sno87, TAO90]). Uma análise mais profunda pode ser encontrada em [MS91],

porém ainda assim restrita a álgebras relacionais estendidas para o suporte a tempo. Em [CC87, TC90] procura-se estabelecer uma noção de completeza relacional histórica.

O primeiro objetivo deste trabalho decorre do grande número de propostas relacionadas a BDT's. A primeira meta desta dissertação é a realização de uma revisão bibliográfica que analise as principais propostas existentes no campo de BDT's.

Embora a pesquisa sobre BDT's seja bastante extensa, a grande maioria das propostas concentra-se em extensões ao modelo relacional. Entretanto, o modelo OO corresponde a uma linha recente de pesquisa com objetivos semelhantes aos modelos temporais — apresentar um esquema que seja capaz de fornecer uma melhor modelagem do mundo real. A necessidade de dotar sistemas OO's das características de um BDT torna-se assim bastante evidente. Os trabalhos existentes neste sentido [CC88, SC91, WD92] são ainda em pequeno número e de modo algum podem ser considerados definitivos. O objetivo principal desta dissertação é contribuir no desenvolvimento desta área, por meio da apresentação de um modelo temporal, baseado no modelo OO, bem como de uma linguagem de consulta para este modelo.

Portanto, as principais contribuições desta dissertação são:

- a realização de uma ampla revisão bibliográfica sobre BDT's;
- a especificação de um modelo que leva em consideração as particularidades do paradigma da orientação a objetos, bem como as características fundamentais da modelagem temporal; e
- a especificação de uma linguagem de consulta para o modelo, que realiza as principais operações de consulta existentes nos modelos analisados na revisão bibliográfica.

Nas seções seguintes descrevemos o modelo OO que servirá de base ao modelo temporal aqui proposto e alguns conceitos temporais fundamentais, que serão largamente utilizados ao longo de toda a dissertação.

1.2 Descrição do Modelo Orientado a Objetos

A área de bancos de dados orientados a objetos (BDOO's) encontra-se sob intensa pesquisa. Em virtude disto, não existe ainda uma definição sobre quais as características essenciais a um BDOO. Entretanto, existe um certo consenso sobre algumas destas características. Baseamo-nos em [ABD+89] para o estabelecimento do nosso modelo base. Nesse trabalho, os autores consideram essenciais em um sistema OO as seguintes características:

- Objetos complexos.
- Identidade de objetos.
- Encapsulamento.
- Tipos e/ou classes.
- Herança e hierarquia de classes.
- *Overriding*, sobreposição (*overloading*) e acoplamento tardio (*late binding*).
- Extensibilidade.

Objetos complexos são objetos construídos a partir de outros objetos mais simples, através de construtores, tais como, listas, conjuntos ou tuplas. A **identidade** de um objeto identifica-o univocamente, distinguindo-o de todos os demais objetos existentes no BD. A idéia essencial da identidade de objetos é a sua existência ser independente de seus valores em um momento qualquer.

O conceito de **encapsulamento** estabelece que os dados dos objetos só podem ser acessados por meio das operações (métodos) predefinidas destes objetos. Ou seja, objetos só podem ser manipulados por meio dos métodos para eles definidos; a composição dos objetos deve ser escondida. Porém, em alguns casos, notoriamente em linguagens de consulta *ad hoc*, a necessidade de encapsulamento é reduzida, e o encapsulamento pode ser violado.

Um **tipo** em um BDOO fornece a representação do estado dos objetos (sua composição) e o conjunto de métodos definidos para estes objetos (seu comportamento). O conceito de **classe**, por sua vez, indica a coleção de todos os objetos (instâncias da classe) que apresentem determinadas características, em geral, possuam o mesmo tipo. Nesta dissertação utilizamos o termo classe como combinação destes dois conceitos.

As classes de um BDOO são organizadas em um grafo (orientado e acíclico) de herança, segundo as relações de especialização existentes entre elas, formando assim uma **hierarquia de classes**. Numa hierarquia de classes, classes mais específicas (subclasses) **herdam** características (métodos e componentes) de classes mais genéricas (superclasses). As características herdadas podem ser aproveitadas integralmente ou redefinidas na subclasse. A redefinição de métodos em diversas classes é chamada *overriding*. Esta redefinição resulta na possibilidade de um mesmo nome denotar diversas operações (o que é denominado **sobreposição**). Para possibilitar a sobreposição de métodos, torna-se necessário que a ligação dos nomes dos métodos com seus códigos seja resolvida em tempo de execução, ao invés de em tempo de compilação. Isto é denominado **acoplamento tardio**.

Extensibilidade designa a capacidade de definir novos tipos a partir dos tipos providos pelo sistema, não existindo distinção na utilização dos tipos originais e os definidos pelo usuário.

Além do grafo de herança, BDOO's normalmente contam com um grafo que especifica as relações de composição existentes entre as diversas classes. Tal grafo é denominado grafo de composição.

1.3 Conceitos Temporais Fundamentais

1.3.1 Dimensões Temporais

A origem da pesquisa em BDT's está fortemente ligada a dois objetivos básicos:

1. A possibilidade de manter estados passados do BD; e
2. A possibilidade de indicar em que momentos os dados presentes no BD foram ou serão válidos no mundo real.

Estes dois objetivos estão relacionados com conceitos de tempo distintos. Modelos que não distingam estes tipos de tempo claramente poderão ter dificuldades em atingir tais objetivos e tenderão tornar confuso o tratamento de certas questões (como atualizações e evolução de esquema). Os primeiros a identificar de maneira clara estes conceitos de tempo foram Snodgrass e Ahn [SA85]. Em [SA86], Snodgrass e Ahn classificam, baseados nestes conceitos de tempo, BDT's em quatro classes com características temporais distintas, de acordo com o suporte que oferecem a estes tipos

de tempo. Em [JCG+92] a terminologia empregada por Snodgrass e Ahn é ligeiramente alterada. Descrevemos abaixo os conceitos introduzidos em [SA85, SA86], já utilizando a nova terminologia.

Snodgrass e Ahn identificam dois tipos básicos de tempo, a saber, tempo de transação e tempo válido. **Tempo de transação** descreve como os dados evoluem com respeito ao BD, isto é, quando foram inseridos, modificados ou excluídos do BD. Este tempo descreve quando as alterações são feitas fisicamente no BD. Os valores relativos ao tempo de transação devem ser supridos automaticamente pelo próprio SGBD e representam os instantes em que as transações que efetuam alterações ao BD são confirmadas (emitem seu *commit*). Por estas características, a semântica do tempo de transação independe da aplicação que está sendo modelada pelo BD. **Tempo válido** descreve como os dados evoluem no mundo real, i.e., quando os fatos geradores das informações armazenadas no BD ocorreram (ou devem ocorrer) no mundo real. Os valores de tempo válido não têm relação necessária com o tempo em que as informações serão registradas no BD e, por conseguinte, poderão diferir dos valores do tempo de transação. Os valores de tempo válido devem, portanto, ser fornecidos pelo usuário.

Tempo válido expressa a visão que o usuário tem do mundo real — um valor de tempo válido associado a um dado do BD indica que o usuário considera que este dado representa uma situação que foi (será) válida no mundo real durante este tempo. Como o tempo válido depende da interpretação do usuário ele pode assumir diversos significados em aplicações distintas, ou mesmo ter mais de uma interpretação para uma mesma aplicação. Suponha o caso de uma transportadora, com serviço próprio de manutenção, que mantenha em um BD os registros dos consertos efetuados em seus veículos. Neste caso, o tempo de fim de conserto poderia ser o do término dos serviços de manutenção, ou o da liberação do veículo pelo departamento de manutenção, ou ainda, o da reintegração do veículo à frota.

Além destes tipos de tempo, Snodgrass e Ahn identificam um terceiro tipo de tempo, **tempo de usuário**, que é simplesmente um tipo de dados cujo domínio é definido sobre valores de tempo. Este tipo de tempo não possui semântica especial para o SGBD, sendo tratado como um tipo de dados comum.

Tempo de transação e tempo válido podem ser interpretados como dimensões de tempo ortogonais, sobre as quais os dados do BD evoluem de forma independente. Por poder receber diversas interpretações, o tempo válido pode determinar mais de uma dimensão temporal, i.e., poderíamos ter uma aplicação com suporte a dois tipos de tempo válido, na qual os dados da aplicação evoluiriam diferentemente em relação a cada um dos tempos. Tempo de transação, ao contrário, por ter uma semântica fixa, pode determinar apenas uma dimensão temporal. Embora alguns modelos [Ari86, GY88] tenham proposto o suporte a um número arbitrário de dimensões temporais, consideramos que a manutenção de duas dimensões de tempo (uma ligada a tempo de transação e outra a tempo válido) é suficiente para modelar a grande maioria das aplicações práticas em BDT's e adotamos esta estratégia em nosso modelo.

A manutenção de cada tipo de tempo em um BDT visa atingir um daqueles propósitos básicos de BDT's. Um modelo que vise manter os estados passados do BD deve prover suporte a tempo de transação. Um modelo que procure possibilitar a indicação de quando os dados do BD tiveram validade no mundo real deve fornecer suporte a tempo válido. Snodgrass e Ahn [SA86] classificam BD's em quatro tipos, de acordo com os tipos de tempo que o BD suporta. São possíveis quatro tipos de BD's: instantâneo [1], de tempo válido, de tempo de transação e bitemporal.

[1] Em inglês, *snapshot*.

Banco de dados instantâneo (BDI) corresponde ao conceito tradicional de BD — não prevê suporte nem a tempo válido nem a tempo de transação. Como não há possibilidade de se indicar o tempo no qual cada informação é válida no mundo real, este tipo de BD pode expressar apenas que os dados estão presentes ou não no BD. Além disto, qualquer alteração ao BD implica na perda do estado anterior, que é substituído pelo novo estado. Isto impossibilita a recuperação de estados passados do BD.

Os dados retratados por este tipo de BD correspondem, então, à visão corrente dos dados, sem a noção de quando estes foram/serão válidos.

O banco de dados de tempo válido (BDTV) permite a indicação do tempo em que os dados do BD são válidos no mundo real — para cada objeto do BD podem ser mantidos diversos valores, válidos em épocas distintas. Alterações aos dados válidos no presente, passado ou futuro são permitidas neste tipo de BD. Contudo, como não suporta tempo de transação, o BDTV, assim como o BDI, retrata apenas o **conhecimento atual** (versão corrente dos dados). Sempre que qualquer dado é alterado no BD, sua versão antiga é perdida, sendo substituída pela nova. Portanto, o BDTV não é capaz de manter estados passados do BD.

O BDTV pode ser visto como um histórico dos dados, mantido atualizado sempre pelo conhecimento atual.

O banco de dados de tempo de transação (BDTT), ao contrário dos tipos de BD's anteriores, retrata tanto conhecimento atual, como conhecimento passado dos dados. Pela própria semântica do tempo de transação, toda nova informação inserida em um BDTT é ligada ao tempo presente, o que resulta na criação de um novo estado do BD a cada atualização e impede a alteração de dados passados. Isto torna o BDTT apropriado para a manutenção de estados passados do BD. O BDTT não suporta tempo válido e, portanto, não é capaz de especificar o tempo em que os dados têm validade no mundo real. Neste tipo de BD, sabe-se apenas quando os dados estiveram armazenados no BD, ou seja, o BDTT mantém a história do BD (quando foram inseridos, alterados e excluídos os dados) e não a história do mundo real (quando estes dados valeram/valerão no mundo real).

Podemos visualizar um BDTT como uma seqüência de BDI's, cada um referente a um tempo de transação. Embora os estados (instantâneos) passados possam ser recuperados, eles não podem ser alterados (lembre-se que o tempo de transação se refere a tempo presente e portanto não podemos associar uma nova informação a um tempo passado). A operação característica deste BD é a recuperação de estados passados do BD, i.e., recuperação dos dados do BD da maneira como eles estavam armazenados em algum ponto do passado.

O banco de dados bitemporal (BDBT) oferece suporte tanto a tempo válido quanto a tempo de transação, possuindo as capacidades do BDTV e do BDTT simultaneamente. Toda informação armazenada em um BDBT deve ter associada a si o tempo em que foi introduzida no BD (tempo de transação) e o tempo durante o qual a informação é válida (tempo válido).

Como acontece com o BDTT, no BDBT se retrata o conhecimento atual e passado dos dados. Por ter associado um valor do tempo de transação, toda alteração do estado do BD é executada sobre o estado corrente do BD e provoca a transição para um novo estado, deixando os estados anteriores do BD imutáveis — permitindo acesso a dados como conhecidos no passado. Ao mesmo tempo, é possível indicar quais os instantes de tempo de validade de cada informação contida no BD, ou seja, cada estado do BD corresponde a um histórico dos dados.

Este tipo de BD pode então ser visto como uma seqüência de BDTV's (inalteráveis) indexados pelo tempo de transação. Com esta combinação de conceitos podemos acessar informações sobre o

passado ou futuro, como conhecidas no presente (selecionando-se o último estado histórico armazenado) ou como conhecidas em qualquer momento da história do BD (selecionando-se o estado histórico correspondente ao tempo de transação desejado).

1.3.2 Variação de Valores no Tempo

Os valores de um objeto podem variar de diversas maneiras com o decorrer do tempo. Segev e Shoshani [SS87, SS88] classificam suas *time sequences* (vide o modelo de Segev na seção 2.1.9) em quatro tipos conforme a forma de variação de valores que elas apresentam. Estas formas de variação, que sumarizam as proposta encontradas na literatura, são descritas abaixo:

Variação Escada: Um objeto que obedeça a este tipo de variação mantém seu valor constante durante o período de tempo entre duas atualizações ao seu valor, ou seja, quando se atribui um valor a um objeto não especificando o instante final de sua validade, este objeto permanecerá com valor constante até que um novo valor seja atribuído a ele.

Variação Discreta: Este tipo de variação apresenta duas características: o valor atribuído a um objeto é válido apenas por um instante e não há ligação entre os valores do objeto em pontos diferentes do tempo — logo, não se pode inferir o valor deste objeto em um momento qualquer através de seus valores em momentos anteriores e posteriores. Se em um ponto do tempo não há um valor armazenado para um determinado objeto é simplesmente porque o objeto não possui nenhum valor neste momento, ou seu valor não é conhecido.

Variação Contínua: Utilizada apenas para dados numéricos, este tipo de variação é aplicável quando o valor da grandeza retratada no BD varia continuamente entre dois instantes de tempo. Como na variação discreta, o valor atribuído a um objeto em um instante de tempo específico é válido apenas para este instante. Entretanto, pode-se inferir o valor do objeto em instantes de tempo em que não há valores armazenados, utilizando seus valores conhecidos em outros pontos de tempo e uma função de interpolação contínua.

Variação Definida pelo Usuário: O usuário pode necessitar formas de variação diferentes das formas padrão. Existe então a possibilidade de se definir novos tipos de variação, onde a interpolação de valores é efetuada por meio de uma função definida pelo usuário.

Note-se que a variação contínua pode ser interpretada como um tipo especial de variação definida pelo usuário, já que a função contínua utilizada pelo SGBD para inferir valores poderia ser uma função definida pelo usuário.

Um segundo fato a ser notado é que *os valores dos objetos sempre variam segundo uma função escada em relação ao tempo de transação.* Como este tipo de tempo descreve a história do BD, o valor de um objeto só se altera quando substituído, o que caracteriza a variação escada. Em relação a tempo válido, porém, todas as formas de variação são possíveis.

1.4 Organização da Dissertação

O objetivo da dissertação é discutir os principais problemas existentes em BDT's orientados a objetos (BDTOO's) e apresentar algumas soluções. Para cumprir estes objetivos os capítulos que se seguem têm o seguinte conteúdo:

- o capítulo 2 apresenta uma revisão bibliográfica comparativa sobre modelos e linguagens temporais, apresentando os problemas e lacunas existentes;
- o capítulo 3 propõe um novo modelo para BDTOO's TOODM que estende as propostas existentes e soluciona vários dos problemas apontados no capítulo anterior;
- o capítulo 4 propõe uma linguagem de consulta temporal TOOL para o TOODM, baseada na O2Query; e
- finalmente, o capítulo 5 apresenta conclusões e enumera extensões possíveis para esta dissertação.

Capítulo 2

Revisão Bibliográfica

Este capítulo apresenta uma revisão bibliográfica sobre BDT's, dando ênfase aos aspectos de modelos de dados, álgebras de manipulação dos modelos e linguagens de consulta, em detrimento de aspectos de implementação. Todavia, levando em consideração a vasta bibliografia publicada, este trabalho não tem a pretensão de ser exaustivo em relação ao tema. Ao invés disto pretende fornecer uma visão ampla do assunto, contendo as principais propostas já apresentadas.

Na seção seguinte são expostos vários modelos propostos na literatura. Na seção 2.2 é realizada uma análise comparativa dos modelos aqui apresentados. A seção 2.3 traz os comentários finais sobre a revisão efetuada.

2.1 Propostas para Bancos de Dados Temporais

Nesta seção descrevemos as propostas de BDT's existentes na literatura, mostrando quando for o caso a evolução que os modelos apresentam com o decorrer do tempo. Cada modelo será referenciado pelo nome de um de seus autores. Os modelos são descritos pela ordem cronológica da publicação do primeiro artigo de sua descrição.

2.1.1 Clifford

Clifford e Warren [CW83] estão entre os primeiros pesquisadores a apresentar um modelo de dados temporal. Este modelo sofre diversas alterações em [CT85] e em [CC87], onde sua versão final recebe o nome de *Historical Relational Data Model* (HRDM).

Em [CW83], Clifford e Warren dedicam-se à formalização das construções do modelo histórico através da lógica intensional. São definidos os conceitos de **intensão** e **extensão**[1] de dados; o segundo se refere a valores de dados, enquanto o primeiro é formalizado como uma função de pontos de tempo para valores de dados (extensões). Uma relação histórica é formada pela união das **relações completadas** de cada estado do BD, sendo que cada estado corresponde aos dados correntes do BD em um tempo específico. Uma relação completada é a relação correspondente a um estado de BD, ampliada com as tuplas de todas as entidades que estejam presentes em algum dos demais estados do BD.

Para representar relações históricas em um BD relacional são acrescentados os atributos *STATE*, para denotar o estado do BD ao qual a tupla pertence, e *EXISTS?*, para indicar se a entidade está

[1] Em inglês, *intension* e *extension*, respectivamente.

ou não sendo modelada pelo BD em um determinado estado. Em tuplas onde o valor de *EXISTS?* é falso os atributos não pertencentes à chave são nulos. Os valores destes atributos entre dois estados consecutivos do BD devem ser inferidos utilizando-se uma função de interpolação; no artigo utiliza-se a variação em escada para a interpolação de valores.

Exemplo:

Suponha que exista um esquema $EMP = \{NOME, DEPTO\}$. Considerando que o estado 1 corresponda a janeiro de 1980 e a distância entre dois estados consecutivos um mês, a representação de uma relação R sobre o esquema EMP seria:

STATE	*EXISTS?*	*NOME*	*DEPTO*
1	1	Luís	Vendas
1	0	Rita	⊥
2	1	Luís	Vendas
2	1	Rita	Vendas
3	1	Luís	Vendas
3	1	Rita	Pessoal
4	0	Luís	⊥
4	1	Rita	Pessoal
5	1	Luís	Pessoal
5	1	Rita	Pessoal
6	1	Luís	Pessoal
6	1	Rita	Pessoal
7	1	Luís	Pessoal
7	0	Rita	⊥
8	1	Luís	Pessoal
8	0	Rita	⊥

Esta relação indica que o empregado "Luís" esteve ligado ao departamento "Vendas" durante os estados 1, 2 e 3, desligou-se da empresa durante o estado 4 e, retornando à empresa, esteve ligado ao departamento "Pessoal" entre os estados 5 e 8. Os dados da empregada "Rita" foram acompanhados durante os estados 2 a 6, tempo em que ela esteve vinculada ao departamento de "Pessoal". O símbolo ⊥ indica o valor nulo. Durante o restante deste capítulo utilizaremos este exemplo para ilustrar a representação dos dados temporais em diversos modelos.

□

Enquanto em [CW83] procura-se formalizar a semântica de um BD histórico utilizando-se a lógica, em [CT85] Clifford concentra-se na definição de uma álgebra histórica para um modelo baseado em tempo. No primeiro artigo os autores afirmam que a relação histórica como apresentada é uma idealização e sua implementação direta traria grande redundância nos dados. No novo modelo os atributos são classificados em três tipos: CA's (*constant attributes*), atributos que não variam durante o tempo, TVA's (*time-varying attributes*), atributos variantes no tempo, representados por

funções de pontos de tempo para valores de dados, e TA's (*temporal attributes*), que representam valores de tempo. A cada relação é associado um *lifespan*, que indica o conjunto de um ou mais períodos de tempo disjuntos em que a relação modela os dados do mundo real. São criados três tipos de valores nulos para expressar valores desconhecidos ou não existentes em pontos específicos do tempo ou valores desconhecidos em todos os instantes de um *lifespan*. Para os instantes de tempo dentro do *lifespan*, sem dados explicitamente armazenados, os valores são inferidos segundo uma função de continuidade ou, para dados não interpoláveis, não existem. A única função para interpolação mostrada no artigo é a em escada.

Exemplo:

Na nova representação, a relação R seria representada da seguinte forma:

NOME	*DEPTO*
Luís	1 — Vendas
	1 — Nulo1
	5 — Pessoal
Rita	2 — Pessoal
	7 — Nulo1

O *lifespan* associado à relação R seria, neste caso, $[1...8]$. O valor "Nulo1" indica a não existência de valores em pontos específicos do *lifespan*.

□

Na álgebra descrita para o modelo os operadores de projeção (π) e seleção (σ) são mantidos como no modelo relacional. O predicado de seleção do operador σ pode especificar o valor de um TVA em um tempo específico, em um ou mais intervalos de tempo, em um tempo qualquer durante seu *lifespan* ou designar os valores do TVA em todo o seu *lifespan*. Os operadores de união ($\cup$), diferença ($-$) e produto cartesiano ($\times$) não são definidos. São introduzidos dois operadores para manipular a dimensão temporal, TIME-SLICE (τ) e WHEN (Ω). O operador τ restringe os dados de uma tabela àqueles válidos durante um *lifespan* específico; o sistema deve gerar automaticamente valores nulos adequados para instantes em que os valores dos atributos não possam ser inferidos. O operador Ω tem como argumento uma condição sobre os dados e retorna o conjunto de intervalos sobre os quais a condição é satisfeita.

Clifford e Croker apresentam em [CC87] a versão final do HRDM. O *lifespan* que anteriormente era associado às relações passa a estar ligado aos atributos, descrevendo o(s) período(s) em que a relação modelou cada propriedade das entidades. O *lifespan* da relação pode ser computado pela união dos *lifespans* de todos os atributos e deve ser igual ao *lifespan* de cada um dos atributos chave. A vantagem de se relacionar *lifespans* a atributos é a possibilidade de se representar a evolução do esquema das relações.

Ortogonalmente ao *lifespan* associado aos atributos, cada tupla é marcada com um *lifespan*, que indica os períodos de sua validade. Embora a noção de período de validade de uma tupla já existisse no artigo anterior [CT85], não havia a marcação explícita das tuplas. A associação de

lifespans às tuplas elimina, muitas vezes, a necessidade de valores nulos: na nova definição não são introduzidos os três tipos de valores nulos como anteriormente. Os valores de um atributo em uma tupla são definidos apenas para a interseção do *lifespan* do atributo com o da tupla. São descritos dois tipos de atributos, que expressam funções do domínio do tempo para outros domínios (TD's) ou para o próprio domínio do tempo (TT's).

Exemplo:

A relação R representada na versão final do HRDM ficaria:

NOME	*DEPTO*	*LIFESPAN*
Luís	1 — Vendas	{[1,3],[5,8]}
	5 — Pessoal	
Rita	2 — Pessoal	{[2,6]}

Os atributos *NOME* e *DEPTO* estariam vinculados individualmente ao *lifespan* $[1\ldots8]$.

□

A álgebra histórica é modificada para o novo modelo. Os operadores $\cup$, $\cap$, $-$, $\times$ e π são mantidos com o significado da álgebra relacional. São criados os operadores UNION-MERGE ($\cup o$), INTERSECTION-MERGE ($\cap o$) e DIFFERENCE-MERGE ($- o$), que realizam a operação original indicada por seu nome modificada para lidar com os *lifespans* das tuplas e os valores dependentes de tempo dos atributos. O operador σ é transformado em dois novos operadores, σ_{IF} e σ_{WHEN}. Este retorna tuplas com o *lifespan* restrito aos pontos em que a condição do predicado é satisfeita, enquanto aquele retorna tuplas com o *lifespan* original. No caso de σ_{IF}, além do predicado devem ser especificados o(s) período(s) de tempo sobre o(s) qual(is) a consulta é efetuada e um dos quantificadores $\forall$ ou $\exists$, indicando se a condição deve ser satisfeita em todos os momentos do(s) período(s) em questão ou se pode ser satisfeita em apenas um momento. O operador Ω é mantido como no artigo anterior e o operador τ é dividido em estático ($\tau_{a/}$) e dinâmico ($\tau_{a:A}$). $\tau_{a/}$ restringe o *lifespan* de uma tabela inteira a um *lifespan* fixo enquanto que $\tau_{a:A}$ restringe o *lifespan* de cada tupla ao valor de um atributo do tipo TT na tupla.

2.1.2 Klopprogge

Klopprogge e Lockemann apresentam em [KL83] uma extensão ao modelo ER para o suporte a dados temporais, a qual denominam *Temporal Entity-Relationship Model* — TERM. Os autores introduzem um mecanismo para inferir estados passados que não estão explicitamente armazenados e afirmam que BDT's introduzem a necessidade de lógica ternária.

Primeiramente a noção de estado é generalizada para histórico — um mapeamento de tempos para valores. Para a inferência de estados não armazenados em um histórico, estarão associadas a cada histórico uma função de derivação — a ser usada para estados que possam ser inferidos com certeza — e um conjunto de funções de aproximação — a serem utilizadas na inferência dos outros estados. Os históricos serão criados a nível de atributos e *roles* (papéis das entidades constantes

num relacionamento), que doravante serão denominados componentes, em oposição a entidades e relacionamentos, chamados objetos.

Componentes podem ser constantes ou variáveis no decorrer do tempo. Além dos valores pertencentes ao seu domínio, componentes podem assumir três valores especiais. **Nulo** é utilizado quando o componente é indefinido no momento; **desconhecido** indica que o componente é definido e tem um valor, porém o valor não é conhecido; **incerto** é utilizado quando o valor pode ser nulo ou desconhecido. Para lidar com estes valores os autores utilizam lógica ternária.

A existência dos objetos é indicada em um atributo obrigatório associado a cada objeto. Como os demais, o atributo de existência pode ser constante ou variável. No segundo caso existirá um histórico de existência. Históricos de componentes não podem assumir o valor **incerto** enquanto históricos de existência não podem conter o valor **desconhecido**.

Os autores descrevem vários algoritmos para determinar a existência e o valor de componentes em estados não armazenados, de acordo com as funções de derivação e aproximação associadas ao seu histórico e a lógica ternária. Por *default* a função de derivação é escolhida para a inferência de um valor não conhecido. Entretanto, uma função de aproximação pode ser selecionada pelo usuário para a determinação do estado não armazenado; caso não haja uma função de derivação e o usuário não especifique a função de aproximação, o resultado é um valor **incerto**.

As funções de derivação e aproximação, bem como a especificação de componentes constantes ou variáveis é realizada na definição do esquema do BD. Foi desenvolvida uma linguagem de definição de esquema TERM e um compilador que mapeia uma interface TERM para um SGBD do modelo em rede.

2.1.3 Lum

Em [LDE+84] argumenta-se que é necessária uma abordagem integrada para projetar o suporte ao tempo diretamente dentro do SBD. Deste modo, os autores propõem um modelo básico para este suporte, e em seguida descrevem estruturas e mecanismos para o suporte a um BDT a nível físico.

O modelo descrito no artigo é baseado, como a maioria das demais propostas, no modelo relacional. Os autores reconhecem a existência de dois tipos de tempo, lógico e físico. O tempo físico corresponde ao tempo do relógio interno no momento de armazenamento dos dados e não pode ser modificado. O tempo lógico se relaciona ao tempo do mundo real e pode ser alterado pelo usuário. O tempo físico é usado para marcar todas as ações no BD, e é deixada ao usuário a tarefa de definição de um ou mais parâmetros temporais que corresponderão ao tempo lógico. Assim sendo, o artigo limita-se à implementação de tempo físico. Note-se que o tempo físico corresponde ao conceito de tempo de transação e o tempo lógico tem elementos de tempo válido e de usuário.

Apesar de considerar a alternativa de adicionar dois atributos temporais nas relações para prover suporte ao tempo físico, os autores chegam à conclusão que tal mecanismo é insuficiente. A solução apresentada é projetar a função de processamento de dados históricos diretamente no SGBD. A partir daí, são descritas várias estruturas de dados e estratégias para este propósito. Podemos destacar dentre as estratégias a opção por separar dados correntes de históricos e por colocar os dados históricos numa cadeia, iniciando pelos dados mais recentes.

Os autores adotam a seguinte estratégia para a correção de erros em estados passados. O sistema gera um diferencial que inclui o tempo efetivo em que o dado deveria se tornar válido; o *timestamp* (tempo físico) não deve ser modificado. Contudo, não fica claro como o sistema tratará estes dois tempos, nem o que deve ser feito em caso de várias alterações. Para o tratamento do

tempo futuro são propostas duas alternativas, a saber, a criação de duas cadeias de dados, uma histórica e uma para o futuro, ou o armazenamento de dados futuros em tabelas diferentes com transferências para tabelas correntes com o passar do tempo.

Como último tópico, os autores abordam a questão de evolução de esquema. Para solucionar este problema, as tabelas de catálogo do sistema devem ser projetadas como as tabelas de dados. Quando ocorressem mudanças no esquema, a informação histórica correspondente seria criada no catálogo. Para interpretar os dados corretamente, deve-se criar a estrutura de dados correspondente, no histórico do catálogo, ao tempo em que os dados foram armazenados.

2.1.4 Snodgrass

Em [SA85, SA86] Snodgrass e Ahn propõem uma taxonomia para unificar a terminologia utilizada para classificação dos tipos de tempo e tipos de bancos de dados deles resultantes. Esta taxonomia é alterada em [JCG+92], onde recebe sua versão definitiva. Introduzimos estes conceitos no capítulo 1 (seção 1.3.1). Utilizaremos a nova terminologia para descrever este modelo, apesar da nova terminologia ter sido definida posteriormente aos artigos do modelo.

Apoiado nos conceitos apresentados na seção 1.3.1, Snodgrass define em [Sno87] a linguagem de consulta TQuel, para manipulação de um BDT construído sobre o modelo relacional. O autor apresenta dois tipos de relações temporais: relações evento, que modelam ocorrências instantâneas, e relações intervalo, que armazenam um estado válido durante um período de tempo. O modelo base para a linguagem suporta relações instantâneas, de tempo válido, de tempo de transação e bitemporais. Para representar estas relações no modelo relacional foram acrescentados atributos temporais implícitos que marcam cada tupla da relação. Para relações com suporte a tempo de transação foram adicionados os atributos *START* quando a tupla foi inserida no BD e *STOP* quando a perdeu sua validade, por atualização ou exclusão. Para relações de tempo válido ou bitemporais é adicionado o atributo *AT* para relações evento ou os atributos *FROM* e *TO* para relações intervalo que indicam o tempo em que os dados da tupla foram válidos no mundo real.

Exemplo:

Retornando ao nosso exemplo, a relação R, representada como uma relação bitemporal, ficaria:

NOME	*DEPTO*	*FROM*	*TO*	*START*	*STOP*
Luís	Vendas	1	3	t1	NOW
Luís	Pessoal	5	8	t2	NOW
Rita	Pessoal	2	6	t3	NOW

Os tempos t1, t2 e t3 correspondem ao tempo de transação no momento de criação das respectivas tuplas. O valor "NOW" indica que as informações são válidas no tempo presente.

□

A linguagem TQuel é uma extensão da linguagem Quel do sistema relacional Ingres. Todos comandos Quel são válidos e mantém a semântica, quando o tempo é fixado a um instante, na

linguagem TQuel. São introduzidas três novas cláusulas, para lidar com as novas dimensões temporais. A cláusula WHEN, análoga à WHERE do Quel, serve para seleção de tuplas de acordo com um predicado temporal. Para a operação de recuperação de um estado passado do BD (denominada *rollback*) é adicionada a cláusula AS-OF. A cláusula VALID é utilizada para especificar o valor do tempo válido para a relação resultante de uma consulta ou para alterações. Consistentemente com as definições anteriores, as alterações são executadas sobre o estado corrente, seu o uso de AS-OF, gerando para relações de tempo de transação e bitemporais um novo estado ou, para os outros dois tipos, modificando o estado corrente. Além destas cláusulas são incluídos os construtores BEGIN-OF, END-OF, OVERLAP (interseção), EXTEND (união) e os operadores de comparação PRECEDE, OVERLAP e EQUAL.

O significado dos comandos TQuel é formalizado pela sua tradução para cálculo de tupla sobre as relações convencionais que são usadas para embutir as relações temporais.

Uma implementação do modelo acima foi realizada estendendo-se o SGBD Ingres. Um *benchmark* executado sobre o protótipo [AS86] demonstrou a necessidade de novos métodos de acesso e estruturas de armazenamento para melhorar o desempenho e os requerimentos de espaço para tornar BDT's viáveis na prática.

A álgebra relacional não considera o fator tempo — opera sobre o estado corrente e o destino das relações derivadas não é considerado. McKenzie e Snodgrass [MS87] propõem a extensão da álgebra relacional tradicional com a introdução dos operadores $\rho/\hat{\rho}$, que correspondem à operação *rollback* sobre uma relação bitemporal ou de tempo de transação, e a criação de uma linguagem de comandos, que teria a álgebra estendida como componente, para providenciar alterações ao estado do BD. São mencionados ainda os operadores $\hat{\cup}, \hat{-}, \hat{\times}, \hat{\sigma}$ e $\hat{\pi}$, que são os equivalentes históricos dos operadores tradicionais. Todavia estes operadores não são descritos no artigo.

O artigo visa primordialmente tempo de transação, estendendo uma álgebra instantânea para uma de tempo de transação. A mesma abordagem pode ser utilizada para estender uma álgebra de tempo válido para uma bitemporal. Um estado de BD é modelado como uma seqüência de estados indexados por tempo de transação. A evolução de estados de um BD é formalizada através da definição do comando MODIFY-STATE (que troca ou adiciona um novo estado ao BD). A formalização destes dois conceitos (estado de BD e evolução de estados) é a principal contribuição do artigo.

Em [MS90] os autores ampliam esta abordagem para suportar o conceito de evolução de esquema. Assim, cada estado de uma relação englobaria seu conteúdo (dados), sua classe (instantânea, de tempo de transação, de tempo válido, bitemporal ou não definida) e sua assinatura (esquema). Os autores definem formalmente uma linguagem algébrica — com a introdução de comandos de alteração do conteúdo, classe e assinatura de uma relação — para consulta e atualização de um BD que suporta evolução e versionamento de esquema e conteúdo.

2.1.5 Tansel

Tansel apresenta em [CT85, Tan86], uma extensão ao modelo relacional para o suporte temporal, com a adição de tempo a nível de atributo, utilizando para isto relações não normalizadas (*non-first normal form*). O autor argumenta que com esta estratégia consegue-se uma abstração mais próxima do mundo real, pois cada tupla pode representar um objeto em toda sua existência. Além disto, elimina-se a redundância gerada nos modelos com *timestamp* a nível de tuplas, que, muitas vezes, precisam repetir valores de atributos não alterados em diversas tuplas.

A razão para marcar o tempo a nível de atributo é que as alterações num BD geralmente

modificam poucos atributos numa tupla. Ocorre uma exceção a este mecanismo no caso de exclusão de tuplas. O autor considera neste caso apropriada a adição de um atributo a todas as relações temporais para armazenar o tempo de exclusão da tupla (na realidade para armazenar o período de validade da tupla).

O tempo adicionado aos atributos está na forma de intervalos, e assim cada valor é representado por uma tripla <[início,fim),valor>. Tansel propõe quatro tipos de atributos para o modelo. Atributos atômicos contêm apenas um valor, constante, não associado a tempo de validade. Um atributo *triplet-valued* também contém um único valor, porém associado a um período de tempo. Um atributo *set-valued* pode conter vários valores atômicos, isto é, não associados ao tempo. Por fim, atributos *set-triplet-valued* podem conter vários valores, cada um associado a um período de tempo. Observe-se que, neste caso, os períodos de tempo podem ou não ser disjuntos.

Exemplo:

Na relação R o atributo *NOME* seria classificado como atômico enquanto o atributo *DEPTO* seria *set-triplet-valued*:

NOME	*DEPTO*
Luís	{<[1,1),Vendas> <[5,9),Pessoal>}
Rita	{<[2,7),Pessoal>}

□

No mesmo artigo Tansel apresenta uma álgebra histórica para o modelo. Os operadores relacionais ($\cup$, $-$, $\times$, π e σ) são deixados praticamente intactos: σ é ampliado para lidar com os diversos tipos de atributos; $\cup$ e $-$ são ampliados para manusear tuplas com os mesmos valores e intervalos que se intersectem.

Novos operadores são introduzidos para o manuseio da dimensão temporal. O operador PACK, sobre um atributo A de uma relação, junta em uma única tupla as tuplas cuja diferença resida apenas nos valores do atributo A. Caso A seja um atributo atômico (*triplet-valued*), poderá se transformar em *set-valued* (*set-triplet-valued*). A operação UNPACK sobre um atributo *set-valued* ou *set-triplet-valued* cria uma tupla para cada valor do atributo na tupla original, repetindo os valores dos outros atributos. O operador TRIPLET-DECOMPOSITION decompõe um atributo (*set-*) *triplet-valued* A em três atributos início, fim e A enquanto que o operador TRIPLET-FORMATION realiza a operação inversa. Além destes, são criados outros operadores não primitivos, ou seja, expressáveis por meio dos anteriores. DROP-TIME descarta a parte temporal de um atributo variante no tempo. SLICE, DSLICE e DSLICE recomputam os *timestamps* de um atributo de acordo com a sua interseção, união e diferença, respectivamente, com os *timestamps* de outro atributo.

Tansel e Garnett em [TG87] estendem o modelo relacional aninhado para suporte temporal, baseando-se no modelo descrito acima, com algumas extensões.

A construção básica do modelo passa a ser um átomo temporal <t,v>. Diferentemente do modelo anterior, t deixa de ser um intervalo temporal para se tornar um conjunto temporal — um

conjunto de intervalos temporais disjuntos. O valor de um atributo qualquer pode ser constituído por um ou mais átomos temporais. Atributos atômicos são também permitidos no modelo.

Exemplo:

No modelo estendido a relação R seria representada como:

NOME	D
	DEPTO
<{[1,4),[5,9)},Luís>	<{[1,4)},Vendas>
	<{[5,9)},Pessoal>
<{[2,7)},Rita>	<{[2,7)},Pessoal>

□

Os atributos de uma relação histórica são organizados hierarquicamente — não há mais o limite de um nível de aninhamento como no modelo inicial — no que é chamado de árvore de esquema. As folhas desta árvore correspondem aos atributos que carregarão os *timestamps* associados aos valores.

Os autores redefinem a álgebra para o novo modelo. Os operadores $\cup$, $-$, $\times$, π e σ são estendidos para efetuar a junção de conjuntos temporais. São criados diversos novos operadores. UNNNEST troca um atributo de nível mais alto na árvore de esquema por seus descendentes; é similar ao UNPACK da álgebra anterior. O operador NEST corresponde ao operador PACK acima apresentado e efetua a operação inversa do operador UNNEST. O operador TRANFER-TIME troca o conjunto temporal de um atributo pelo de outro. Os operadores SLICE, USLICE e DSLICE são mantidos.

Finalmente, em [TAO90] é apresentada a linguagem gráfica TBE — *Time By Example*. A linguagem, segundo os autores, apresenta o mesmo poder de expressividade que a álgebra histórica introduzida em [Tan86]. Os autores descrevem ainda uma metodologia para converter as consultas da linguagem numa árvore de derivação, que auxilia no processo de otimização de consulta.

2.1.6 Gadia

No modelo proposto por Gadia nos artigos [Gad86, Gad88] o tempo é representado por um intervalo fechado T = [0,1,2,...,*now*] e um **elemento temporal**, construção básica do modelo, é um conjunto formado pela união finita de intervalos em T. A partir daí são definidas as outras construções. Uma **atribuição temporal** a um atributo A é uma função que mapeia intervalos de um elemento temporal para valores do domínio de A. Cada atributo numa **tupla temporal** é uma atribuição temporal. Por fim, uma **relação temporal** é um conjunto finito de tuplas temporais. Gadia denomina seu modelo temporal de homogêneo, porque o tempo de validade de todos os atributos numa tupla é igual.

A conseqüência desta representação é que as relações não estão mais na primeira forma normal. Isto acontece porque, apesar do *lifespan* de todos os atributos numa tupla ser o mesmo, podem existir diversos valores associados a cada atributo, cada valor correspondendo a um elemento temporal contido no *lifespan* da tupla.

Exemplo:

A relação R para o modelo de Gadia seria expressa na forma:

NOME	*DEPTO*
[1,4) ∪ [5,9) — Luís	[1,4) — Vendas
	[5,9) — Pessoal
[2,7)— Rita	[2,7) — Vendas

□

Uma seleção temporal restringe o domínio temporal de uma relação a um elemento temporal qualquer. Assim, os dados não concernentes a este elemento temporal são eliminados da relação. Um instantâneo temporal corresponde a uma seleção temporal para um único instante de tempo. Se duas relações contêm instantâneos temporais iguais para todos instantes de tempo em [0,...,*now*], elas são consideradas *weakly equal*. Uma classe de equivalência contém relações *weakly equals* e é denominada *weak relation*. Uma tupla pertence fracamente a uma relação R, se ela pertence a alguma relação da *weak relation* à qual R pertence. Com base nestes conceitos, Gadia introduz em [Gad88] uma álgebra e um cálculo relacional de tupla para o modelo.

Na álgebra do modelo existem expressões relacionais e temporais; as primeiras representam relações temporais enquanto expressões do segundo tipo representam elementos temporais. Entre elementos temporais são permitidos os operadores padrão de conjunto ($\sim$, $\cup$, $\cap$ e $-$). Os operadores relacionais tradicionais ($\cup, -, \times, \sigma$ e π) são redefinidos para garantirem a homogeneidade das tuplas resultantes da aplicação destes operadores e lidarem com atribuições temporais. A aplicação de um dos operadores relacionais modificados sobre uma relação histórica é equivalente à aplicação do operador relacional original sobre cada um dos instantes do *lifespan* da relação. São criados ainda dois novos operadores exclusivamente para lidar com a dimensão temporal. O primeiro tem a função de realizar uma seleção temporal e o segundo serve para retornar o elemento temporal de uma relação histórica.

Gadia e Vaishnav apresentam em [GV85] uma linguagem similar à Quel, chamada HTQuel (*Homogeneous Temporal Query Language*), para o modelo acima. Os autores afirmam que HTQuel é ao menos tão poderosa quanto a álgebra e o cálculo introduzidos em [Gad88]. A linguagem possui vários operadores para navegação (FIRSTINTERVAL, PREVIOUSINTERVAL, LASTINSTANT, NEXTINSTANT, etc) que auxiliam no manuseio de elementos temporais. São incluídas ainda as cláusulas DURING, equivalente a uma seleção temporal, e TDOM, que equivale a uma projeção sobre a dimensão temporal, para recuperação de dados. Atualizações são feitas com os comandos CHANGE, APPEND e DELETE. Alterações a estados passados são feitas naturalmente por CHANGE. Os tempos de estados passados podem ser alterados por um DELETE seguido de APPEND. A cláusula DURING é utilizada para determinar o tempo de validade da atualização. Por fim, atualizações para tempos futuros são permitidas, com o uso da cláusula EFFECTIVE, que substitui DURING neste caso. Além destas características HTQuel admite construções como comandos iterativos e condicionais.

Em [GY88], Gadia e Yeung generalizam o modelo para suportar um número arbitrário de dimensões de tempo (até aqui o modelo era limitado a apenas uma dimensão temporal). Primeira-

mente, os autores redefinem o modelo e a álgebra, ainda considerando apenas uma dimensão, sem incluir o conceito de homogeneidade, o que já se constitui numa ampliação do modelo. Após isto, o modelo e a álgebra são generalizados para n dimensões temporais, com os autores mostrando como ilustração o caso particular de duas dimensões temporais (tempo válido e de transação).

2.1.7 Adiba

Os principais objetivos do modelo de Adiba e Quang [AQ86] são assegurar acesso rápido a dados correntes, suportar dados históricos de objetos com diversas granularidades, estender as linguagens de definição e manipulação de dados para lidar com dados históricos e considerar o problema de evolução de esquema. A proposta apresentada para isto é baseada no modelo ER, estendido com o conceito de tipo.

Os dados temporais são mantidos por meio de históricos das versões dos objetos, onde um histórico constitui uma seqüência de pares (valor,tempo). Há três tipos de históricos:

histórico manual , onde o usuário indica explicitamente quando deseja que uma nova versão seja gerada;

histórico periódico , onde novas versões só são criadas ao final de cada período, não importando quantas modificações ocorram durante o período; e

histórico sucessivo , onde toda modificação provoca a geração de nova versão.

Todo histórico tem associado a si uma persistência. A persistência de um histórico indica quantas das (últimas) versões históricas serão mantidas, podendo ser ilimitada ou limitada a um número fixo de versões. Outra característica dos históricos é a granularidade na qual o tempo é armazenado. O tipo de tempo mantido corresponde ao tempo de transação de Snodgrass [SA85, SA86].

Pela introdução de tipos os autores desejam incluir maior semântica na descrição dos objetos do BD. Há diversos tipos de dados, tais como registro, lista, documento — estrutura para objetos multimídia como voz, texto e imagem — além dos tipos básicos — real, inteiro, etc. Entidades, relacionamentos e as noções de generalização, especialização e agregação são denominados tipos de classe e podem possuir atributos. Um tipo é chamado dinâmico quando é definido como histórico, e estático em caso contrário. Um tipo pode ser constituído de vários níveis de construção, e a escolha do nível em que se declarará a dinamicidade é deixada ao usuário. Uma vez declarada, a dinamicidade de um tipo é transmitida a seus componentes, em cascata.

São permitidas as operações de inserção, atualização, exclusão e correção sobre históricos. A inserção gera a versão corrente de um novo histórico. O funcionamento da operação de atualização depende do tipo de histórico. Para histórico sucessivo o novo valor passa a ser o corrente e o valor anterior é copiado para a área histórica; para histórico manual o usuário deve escolher entre atualizar apenas a versão corrente ou, explicitamente, gerar uma nova versão; para histórico periódico a modificação é executada apenas sobre a versão corrente e o sistema copia os dados periodicamente para a área histórica. A exclusão de objetos dinâmicos é executada por meio de sua gravação, na área histórica, com um *flag* de exclusão. A exclusão de um objeto estático implica na remoção física de todos os seus dados, inclusive os históricos de componentes dinâmicos que porventura existam.

Os autores introduzem algumas extensões à linguagem LAMBDA, que por sua vez é baseada no SQL, para a manipulação de históricos. O manuseio dos dados históricos é executado através

da indicação do conjunto de versões desejadas. Esta indicação é efetuada pela posição ordinal das versões ou pela data na qual elas foram criadas.

Para possibilitar acesso eficiente aos dados correntes, os dados históricos são mantidos numa área separada destes. A noção de históricos é aplicada também aos catálogos do BD para dar suporte a evolução de esquema, permitindo o armazenamento de esquemas passados.

2.1.8 Ariav

O TODM (*Temporally Oriented Data Model*), apresentado por Ariav em [Ari86], é uma tentativa de representar dados temporais através de uma construção tridimensional — o cubo de dados. O modo como o BD progrediu durante o tempo pode ser visto como um cubo, e o estado de uma relação em qualquer ponto do tempo pode ser obtido cortando-se o cubo horizontalmente na altura do tempo especificado.

Há dois tipos de atributos temporais no TODM: TRA's (*time related attributes*) e TSA's (*timestamp attributes*). Qualquer atributo que contenha dados de tempo é classificado de TRA. TSA's são um tipo especial de TRA's: se algum atributo de um intervalo é alterado, o TSA associado à entidade precisa ser atualizado. Isto assegura que toda modificação ocorrida numa entidade esteja associada a um valor de tempo diferente. Toda relação tem ao menos um TSA, denominado RT (*registration time*). O RT não é visto pelo usuário, sendo criado e gerenciado pelo sistema, e serve para marcar as tuplas no momento de seu armazenamento no BD — equivale ao tempo de transação de Snodgrass [SA85, SA86].

A unidade de tempo de uma relação é denominada *Chronon* e expressa a granularidade do tempo de uma aplicação. *Chronon* pode corresponder a segundos, horas, meses ou outra unidade de tempo qualquer.

Um evento no ambiente do BD corresponde a uma TAT (*temporally anchored tuple*). TAT's nunca são removidas; em caso de modificação um nova TAT é criada. Toda TAT é automaticamente marcada, na sua criação, com um *timestamp* — este é o tempo armazenado no atributo RT. Uma TAT é uma tupla de uma entidade em relação a um tempo específico.

Um cubo de dados é uma extensão cúbica de uma relação R para um TSA T (denota-se $E^c(R,T)$). Uma relação pode constituir vários cubos de dados, um para cada TSA existente no seu esquema, inclusive para o RT. O cubo representa como as entidades de uma relação evoluíram em relação ao tempo do TSA; no caso do RT, representa como evoluíram em relação ao tempo de transação, ou seja, é a história do BD.

Exemplo:

A relação R seria representada no modelo de Ariav da seguinte forma:

NOME	*DEPTO*	*TEMPO*	*RT*
Luís	Vendas	1	t1
Luís	Nulo	4	t2
Luís	Pessoal	5	t3
Luís	Nulo	9	t4
Rita	Vendas	2	t5
Rita	Nulo	7	t6

Os atributos *TEMPO* e *RT* são os TSA's da relação; os tempos t1 a t6 correspondem aos valores do tempo de transação no instante de criação das respectivas tuplas. Ariav não explicita como é indicada a não existência das entidades em alguns instantes do tempo; para o exemplo supusemos o uso de valores nulos.

□

As operações primárias no TODM correspondem à extração de cubos menores de um cubo de dados, fazendo restrições em uma de suas três dimensões. A projeção serve para indicar os atributos a serem extraídos de um cubo (note-se que o TSA que determina o cubo não é considerado um atributo). Uma seleção de objetos é utilizada para restringir os objetos que estarão contidos no cubo extraído. Finalmente, uma seleção temporal delimita as fronteiras superior e inferior do cubo extraído.

A partir de um subconjunto do SQL, Ariav acrescenta algumas extensões para lidar com o aspecto temporal, definindo a linguagem de consulta TOSQL. Os *defaults* fazem com que consultas que não especifiquem a parte temporal tenham o mesmo significado que no SQL. Um comando TOSQL tem a seguinte forma:

```
SELECT ...
FROM ...
WHERE ...
{AT | WHILE | DURING | BEFORE | AFTER} ...ALONG ...
AS-OF ...ALONG ...
```

A cláusula SELECT corresponde à projeção; FROM tem o objetivo de determinar qual será a relação do cubo de dados (note-se que apenas uma relação pode ser designada); a cláusula WHERE corresponde à seleção de objetos, enquanto que as cláusulas da quarta linha correspondem à seleção temporal. A cláusula AS-OF tem a função de realizar uma operação *rollback*, como a cláusula homônima em TQuel (veja modelo de Snodgrass [Sno87]), com a diferença que a operação pode ser realizada em qualquer dimensão temporal e não apenas sobre tempo de transação. O significado preciso da operação *rollback* no modelo não é definido, sendo que esta operação não é citada entre as operações básicas do modelo. A opção ALONG é utilizada para definir a dimensão temporal a ser utilizada na consulta histórica e operação *rollback*.

As operações de correção de erros, apesar de serem possíveis dentro do modelo, não são detalhadas no artigo. Segundo Ariav, a estrutura do TODM também está apta a acomodar tempo futuro, mas afirma que o suporte a tempo futuro precisa ser melhor examinado e formalizado.

2.1.9 Segev

A abordagem escolhida neste modelo difere da utilizada nos outros trabalhos em BDT's. Ao invés de estender um modelo de dados já existente para o suporte a dados temporais, os autores buscam determinar a semântica deste tipo de dado e independentemente de qualquer outro modelo, propõem um modelo de dados temporal (*Temporal Data Model* — TDM). O principal conceito do TDM é o de *time sequence* (TS), que designa uma coleção de valores de dados assumidos por uma instância de uma entidade em diversos momentos do tempo. Em [SK86] Shoshani e Kawagoe introduzem os principais conceitos do modelo. Um valor temporal é indicado pela tripla <s,t,v>, onde **s** (*surrogate*) é um identificador único de uma entidade, **t** o tempo no qual o valor foi atribuído

à entidade e **v** o valor assumido. Uma TS é indicada por <s,(t,v)*> e representa uma série de pares (tempo,valor) para uma determinada característica de uma entidade identificado por **s**. Neste artigo os autores descrevem várias propriedades das TS's, tipos de operações possíveis sobre estas e estruturas físicas para o armazenamento destas.

Num artigo posterior [SS87], Segev e Shoshani refinam o modelo, introduzindo novos conceitos, definindo melhor as propriedades das TS's e as operações sobre estas. Neste segundo artigo são definidas as seguintes propriedades das TS's:

Granularidade de tempo: Pode ser ordinal — uma seqüência de pontos numerados (1,2,3,...) — ou de calendário — ano, mês, dia, hora, etc.

Life Span: Intervalo de validade da TS, determinado pelos pontos de início e fim. Há três casos possíveis: quando os pontos de início e fim são fixos; quando apenas o ponto de início é fixo e o ponto final corresponde ao tempo presente; e quando existe uma distância fixa do ponto inicial ao final, o ponto final corresponde ao tempo presente e o tempo inicial varia.

Regularidade: Uma TS é dita regular quando para cada ponto de tempo de seu *life span* há um dado armazenado, caso contrário a TS é dita irregular. Os pontos de tempo que contêm dados são denominados pontos de dados ou pontos evento.

Tipo: O tipo da TS determina os valores do atributo associado à TS para os pontos de tempo que não contém dados explicitamente armazenados. Em geral há uma função de interpolação associada a cada TS. Os quatro tipos de TS's correspondem aos tipos de variação descritos no capítulo 1 (seção 1.3.2).

Os autores definem como estrutura básica do TDM a *time sequence collection* (TSC), que consiste numa coleção de TS's de mesma classe e com as mesmas propriedades. Uma classe é uma coleção de objetos com os mesmos atributos. As TSC's são classificadas como simples — quando contêm um único *surrogate*, tempo e atributo em cada valor temporal de suas TS's — ou complexas — quando pode haver mais de um identificador, mais de um tempo e/ou mais de um atributo associados à TS. Note-se que no caso de mais de um tempo, várias dimensões temporais poderiam ser incorporadas ao TDM. Entretanto, os autores não voltam mais ao tema e, num artigo posterior [SS88], referenciam como complexas apenas TSC's que estejam representando diversos atributos, não mencionando os outros dois casos.

Os autores definem em [SS87] cinco operadores básicos e um genérico para manipulação de TSC's. Tais operadores têm três partes bem definidas. A **especificação de destino**, para indicar os pontos de dados da TSC destino, o **mapeamento**, que relaciona quais pontos de dados da(s) TSC(s) de origem serão manipulados para gerar os pontos de dados da TSC de destino, e **função**, que é a função a ser aplicada aos valores fontes para se obter o valor destino.

Em [SS88] os autores discutem como mapear seu modelo para o relacional. Para tanto, é definido o conceito de relação temporal, na qual as propriedade das TSC's, tais como tipo, *life span* e granularidade, são consideradas meta-dados e associadas às relações. Uma TSC é representada na forma tabular por meio de uma relação temporal onde cada tupla será marcada com um instante de tempo. Os valores para tempos não armazenados são inferidos com o auxílio dos meta-dados associados à relação.

É definido o conceito de forma normal temporal para as novas relações temporais. A forma normal consiste em assegurar que para uma dada instância e ponto de tempo, cada atributo só pode ter um valor associado.

Exemplo:

A relação R mapeada para a representação relacional do modelo de Segev seria expressa na forma:

NOME	*TEMPO*	*DEPTO*
Luís	1	Vendas
Luís	4	Nulo
Luís	5	Pessoal
Rita	2	Vendas
Rita	7	Nulo

À relação R deverá estar associado o *life span* [1..8] e a função de interpolação em escada. De modo similar ao que fizemos com o modelo de Ariav, supusemos que a inexistência de uma entidade em momentos contidos no *life span* da relação é expressa por meio de valores nulos.

□

O conceito de famílias de TSC's é definido para se poder visualizar todos os dados de um *surrogate* como uma única unidade. Uma família é uma coleção de relações, temporais ou não, com o mesmo *surrogate* e provê um modo conciso de especificar condições que se aplicam a múltiplas TSC's.

A integração das capacidades de uma linguagem de consulta tradicional com os conceitos do TDM ainda não foi realizada no modelo e é citada como trabalho futuro.

Gunadhi e Segev apresentam em [GS90] uma análise de questões de otimização de consultas em BDT's. A motivação é estudar a possibilidade de implementação do TDM na forma relacional. O artigo concentra-se no estudo de JOINS temporais e no mensuração e estimação de seletividade em relações temporais. O artigo mostra ainda a implementação e otimização de um tipo de JOIN temporal.

2.1.10 Navathe

Navathe e Ahmed [NA89] estendem o modelo relacional para dar suporte à dimensão temporal. Os autores do modelo, denominado *Temporal Relational Model* (TRM), consideram essencial a manutenção da primeira forma normal, isto é, a associação dos *timestamps* às tuplas, e criam a forma normal de tempo (*time normal form – TNF*) para eliminar as deficiências desta estratégia. Inicialmente os autores classificam os atributos variantes no tempo de uma relação em síncronos e assíncronos. Dois atributos são ditos síncronos se variam ao mesmo tempo, ou seja, sempre que o valor de um dos atributos é alterado, o valor do outro também sofre modificação. A normalização de uma relação variante no tempo (*time varying relation – TVR*) consiste em dividir seus atributos em atributos não variantes no tempo e grupos de atributos síncronos e, em seguida, dividir a TVR em relações contendo os atributos da chave primária e, em cada uma, um dos conjuntos de atributos da relação original. A utilização da TNF evitaria, segundo os autores, a redundância de dados e algumas anomalias na recuperação e atualização características de propostas que mantêm a primeira forma normal.

O TRM efetua a marcação de tempo com o uso de intervalos e suporta, por *default*, tempo válido. Os autores afirmam que a dimensão de tempo de transação poderia ser igualmente suportada no modelo, se o usuário assim o solicitasse. Entretanto, não definem operações que utilizem este tipo de tempo.

Exemplo:

Representando a relação R neste modelo teríamos:

NOME	*DEPTO*	T_S	T_E
Luís	Vendas	1	3
Luís	Pessoal	5	8
Rita	Vendas	2	6

□

Os autores apresentam a linguagem de consulta TSQL para a recuperação de informações em BD's baseados no modelo. A linguagem é uma extensão do SQL em que todos os comandos originais são válidos e mantêm o mesmo significado na ausência de referência a tempo. São introduzidas algumas novas construções para manipulação da dimensão temporal. A cláusula WHEN é utilizada para avaliar predicados temporais relativos aos períodos de validade das tuplas. Os *timestamps* associados às tuplas podem ser acessados por meio dos sufixos .INTERVAL, .TIME-START e .TIME-END. São incluídos vários comparadores de intervalos (BEFORE, AFTER, DURING, EQUIVALENT, ...) para formação de predicados temporais. A cláusula TIME-SLICE tem a função de restringir os dados a serem considerados na consulta aos válidos durante um período de tempo específico. A linguagem possui ainda a capacidade de ordenar de forma ascendente as tuplas de uma entidade pelos seus períodos de validade para posterior recuperação e de utilizar as funções de agregação (COUNT, SUM, MIN, ...) sobre a duração da validade das tuplas.

Embora sejam definidos três operadores algébricos para atuarem sobre TVR's (dois tipos de JOINS temporais e um operador para reunir tuplas de valores iguais válidas por períodos de tempo adjacentes), não é apresentada uma álgebra consistente para a manipulação das estruturas do modelo.

Em [MNA87] Martin, Navathe e Ahmed propõem um mecanismo para lidar com o problema da evolução de esquema. A abordagem tradicional de armazenamento de esquemas anteriores não é considerada suficiente, pois obriga o usuário a conhecer todos os esquemas que já existiram para fazer alguns tipos de consulta. Para solucionar esta questão os autores escolhem uma abordagem lógica.

É desenvolvida uma lógica temporal modal, chamada lógica temporal de esquema (*Schema Temporal Logic* - STL), que será usada para traduzir consultas contra o esquema corrente do BD para consultas equivalentes nos esquemas prévios. Cada esquema é associado a um conjunto de fórmulas STL que o descrevem. Cláusulas base descrevem as relações contidas no esquema e sentenças indicam relações contidas em outros esquemas.

Numa consulta, as relações envolvidas são verificadas cotejando as cláusulas base e, caso as relações descritas por estas não possam responder à consulta completamente, verificando-se as sentenças, que poderão enviar a consulta a esquemas prévios, onde o processo é repetido.

Uma reorganização de esquema contém os seguintes passos. Primeiro, as relações que não se alteraram passam para o novo esquema (dados e fórmulas). Segundo, para as relações que se alteraram os dados e fórmulas são mantidos no velho esquema. Finalmente, as "novas" relações e as sentenças indicando relações que se mantiveram no esquema anterior são introduzidas no novo esquema.

2.1.11 Abbod

Como a maioria dos outros pesquisadores, Abbod, Brown e Noble estendem em [ABN87] o modelo relacional para o suporte a dados temporais, através do registro de tempo a nível de tuplas. São considerados dois tipos de tempo, lógico e físico (correspondem a tempo válido e de transação). Os autores reconhecem dois tipos de atualizações ao BD, que são tratadas diferentemente. Atualizações de versão são atualizações que correspondem a alterações no mundo real, e atualizações de correção que, como o nome indica, são para correções de erros detectados no BD.

A marcação de tempo das tuplas é feito com pontos de tempo (em oposição a intervalos). A razão para esta opção é a possibilidade de uso de disco laser *write once* em sistemas de BDT's. No caso de registro de tempo por intervalos não se conhece o valor do tempo final no momento de criação da tupla, o que,segundo os autores, inviabilizaria este tipo de dispositivo. Para auxiliar a manipulação dos dados históricos, adiciona-se também um número de versão a cada tupla, que indica a posição ordinal da tupla na seqüência de tuplas de uma entidade.

O esquema das relações é aumentado em seis atributos: S (*surrogate* – identificador do objeto), T (tempo lógico), P (tempo físico), V (número de versão), C (número para correção) e E (explicação). Estes atributos são meta-dados sob controle do sistema, enquanto os valores de T e E são fornecidos pelo usuário.

Os atributos S, T, P e V correspondem a conceitos já introduzidos. A principal função do atributo E é indicar quais atributos tiveram seus valores modificados para a geração de uma nova versão. O atributo C tem como finalidade possibilitar a execução de atualizações de correção. As versões criadas por meio de atualização de versão têm o valor de C igual a zero. Quando ocorre uma atualização de correção numa versão qualquer de uma tupla, é criada uma nova versão com o mesmo valor de V e com C igual ao número correções feitas à tupla. Além disto, a correção é executada também nas versões seguintes daquela tupla que não tenham sido gerados por outra modificação do mesmo atributo.

Exemplo:

Com os atributos adicionais a representação da relação R ficaria:

S	*T*	*P*	*V*	*C*	*E*	*NOME*	*DEPTO*
#1	1	t1	1	0	Nulo	Luís	Vendas
#1	4	t2	2	0	*NOME*, *DEPTO*	Nulo	Nulo
#1	5	t3	3	0	*NOME*, *DEPTO*	Luís	Pessoal
#2	2	t4	1	0	Nulo	Rita	Pessoal
#2	7	t5	2	0	*NOME*, *DEPTO*	Nulo	Nulo

Como foi feito nos modelos de Ariav e de Segev, utilizamos valores nulos para representar a inexistência das entidades.

□

Um ponto original abordado pelos autores é o do tratamento de restrições de integridade. Os autores argumentam que seria inapropriado aplicar restrições atuais a dados antigos no momento de se efetuarem correções. A estratégia adotada para solucionar esta questão é o armazenamento de restrições antigas com seu período de validade. Tais restrições não estariam sujeitas a atualizações de correção; ao invés disto estão sujeitas a um conjunto de meta-restrições que impediriam o usuário de definir restrições incoerentes.

Os autores relatam a implementação de um primeiro protótipo do modelo, chamado SISBASE, como um *front-end* temporal para o SGBD Ingres. O protótipo não cobre todas as características do modelo e um segundo protótipo já estaria em implementação segundo os autores.

2.1.12 Lorentzos

A abordagem utilizada por Lorentzos e Johnson em [LJ88] é a de mostrar a incapacidade da álgebra relacional convencional em manipular dados temporais e propor uma álgebra relacional temporal (*Temporal Relational Algebra* — TRA) para solucionar este problema.

O modelo desenvolvido pelos autores marca tuplas com tempo, preservando a primeira forma normal e delega ao usuário a escolha entre pontos ou intervalos de tempo para marcação das tuplas.

Os autores tratam tempo e sua granularidade de maneira diferenciada das outras propostas. Um **domínio de tempo** (DT) é um conjunto não vazio, finito e totalmente ordenado de elementos do mesmo tipo. Este "tipo" pode ser segundos, dias, meses, anos ou qualquer outro, com a definição dos elementos inicial e final. O elemento final de um DT não precisa ser fixo: pode ser o tempo corrente ou um tempo futuro a uma distância fixa do tempo corrente. Esta característica, segundo os autores, permitiria o uso de tempo futuro no modelo.

Exemplo:

Representando a relação *R* teríamos:

NOME	*DEPTO*	*MÊS-INÍCIO*	*MÊS-FIM*
Luís	Vendas	1	3
Luís	Pessoal	5	8
Rita	Pessoal	2	6

□

A álgebra proposta pelos autores incorpora os operadores relacionais σ, π, $\cup$, $-$, $\times$ e $\bowtie$ com as características originais. Da mesma forma são incluídos os operadores RENAME (para renomear atributos numa consulta) e COMPUTE (para possibilitar a incorporação de funções). Para lidar com dados temporais são incorporados três operadores. FOLD converte uma relação com tuplas marcadas com pontos de tempo para uma equivalente com intervalos. UNFOLD executa a operação inversa, ou seja, transforma cada tupla numa relação marcada com intervalos num conjunto de tuplas, uma para cada instante do intervalo de validade da tupla original. O operador EXTEND é similar ao UNFOLD, exceto que os atributos de tempo dos intervalos da relação original não desaparecem na relação resultante.

Exemplo:

A aplicação do operador UNFOLD sobre os atributos *MÊS-INÍCIO* e *MÊS-FIM* da relação *R* resultaria na seguinte relação:

NOME	*DEPTO*	*MÊS*
Luís	Vendas	1
Luís	Vendas	2
Luís	Vendas	3
Luís	Pessoal	5
Luís	Pessoal	6
Luís	Pessoal	7
Luís	Pessoal	8
Rita	Pessoal	2
Rita	Pessoal	3
Rita	Pessoal	4
Rita	Pessoal	5
Rita	Pessoal	6

□

Uma característica interessante do modelo é que o resultado de um produto cartesiano (ou de um JOIN) de duas relações *folded* (marcadas com intervalos) é uma relação *twice folded*, em que os dois registros de tempo estão presentes. Neste caso, os DT's podem ser diferentes e a relação resultante pode servir para modelar eventos periódicos.

O uso do operador COMPUTE permite a conversão entre DT's arbitrários e a simulação funções de interpolação diferentes da função escada.

O modelo foi implementado usando-se o SGBD Ingres para gerenciamento de dados de baixo nível.

2.1.13 Sarda

O modelo de Sarda é apresentado primeiramente em [Sar90a] e reapresentado com alguns refinamentos em [Sar90b]. Assim, sempre que houver diferenças entre os artigos, será dada preferência à solução do mais recente.

O conceito fundamental do modelo é o de estado de uma entidade. Enquanto uma tupla numa relação convencional representa uma entidade, numa relação histórica cada tupla representa o estado de uma entidade durante um período de tempo. Isto é conseguido associando a cada tupla um intervalo de tempo, para o qual o estado é válido.

Para o autor, o tempo do mundo real (válido) é a dimensão temporal mais importante, e, geralmente, é igual ao tempo de transação. Por isto, o modelo é voltado para o suporte a uma só dimensão de tempo e o valor adotado por *default* para o tempo é o de transação. No entanto, existe a possibilidade de adequar o tempo adotado para tempo válido , quando este diferir do tempo de transação. Pode-se ainda requisitar o suporte a tempo de transação. Além disto, existe o tipo de dados tempo, que pode ser usado para suporte a tempo de usuário.

São ainda características do modelo: a separação de dados correntes e históricos feita de modo quase transparente ao usuário; a possibilidade de selecionar a granularidade dos valores de tempo e a existência de relações evento (além das relações intervalo), que correspondem a estados válidos para uma única unidade de tempo.

Exemplo:

A relação R seria representada neste modelo da seguinte forma:

NOME	*DEPTO*	*FROM*	*TO*
Luís	Vendas	1	3
Luís	Pessoal	5	8
Rita	Pessoal	2	6

□

Sarda define uma álgebra relacional histórica para o modelo. Inicialmente, relações históricas são vistas como relações normais com dois atributos implícitos adicionais (*FROM*, *TO*) para representação do período de validade da tupla. Assim, os operadores da álgebra relacional convencional ($\cup$, $-$, $\times$, σ e π) são aplicáveis com o mesmo significado; no entanto, os resultados destas operações nem sempre são novas relações históricas. São incluídos dois novos operadores primitivos (EXPAND e COALESCE) e um auxiliar (produto concorrente) para lidar com a semântica temporal das novas relações. O operador EXPAND faz com que cada tupla válida para um intervalo seja transformada num conjunto de tuplas, uma para cada instante do intervalo (análogo ao operador UNFOLD de

Lorentzos [LJ88]). O operador COALESCE combina em uma única tupla as tuplas que têm os mesmos dados e intervalos de tempos consecutivos ou com interseção. O período de validade da tupla resultante será a união destes intervalos. O produto concorrente é similar ao produto cartesiano, todavia emparelha tuplas apenas quando a interseção dos intervalos de tempo destas é não vazio. O intervalo de tempo da tupla resultante será esta interseção.

A linguagem de consulta definida para o modelo, chamada HSQL, é uma extensão do SQL que suporta operações e comparações sobre instantes e intervalos de tempo, atributos de tempo explícitos, bem como os implícitos *FROM* e *TO*, e os operadores da álgebra acima descritos. Para definição de dados a declaração CREATE é aumentada para especificar a granularidade de tempo e determinar se a relação será de estados ou de eventos. Para este segundo tipo de relação os atributos *FROM* e *TO* sempre conterão o mesmo valor.

A declaração para recuperação de dados é ampliada com várias cláusulas. FROMTIME ... TOTIME ... restringe os dados retratados pela relação (ou relações) considerada àqueles válidos durante o tempo especificado. EXPAND BY equivale ao operador algébrico EXPAND. A cláusula FROM é estendida com a opção CONCURRENT, que altera o seu significado de produto cartesiano para produto concorrente. A cláusula SELECT é ampliada com a opção COALESCED, com o mesmo sentido do operador COALESCE.

As operações de atualização normais não têm sua sintaxe alterada. O sistema fornece automaticamente o valor de tempo de transação para ser usado como tempo válido. Para se realizar atualizações retroativas ou pré-ativas inclui-se a cláusula FROMTIME ... TOTIME ... (para relações intervalo) ou a cláusula AT ... (para relações evento) na declaração de atualização.

Para o suporte a tempo de transação o seguinte esquema é adotado. Na definição da tabela inclui-se a cláusula WITH ROLLBACK FACILITY. O SGBD mantém então um "*log* de correção" para a tabela, para os casos em que o tempo válido difere do tempo de transação. Pode-se utilizar então a declaração ROLLBACK ... AS OF ... para se recuperar uma relação como conhecida em algum momento do passado.

2.1.14 Käfer e Käfer$_2$

O modelo de dados temporal descrito em [KRS90] tem como fundamento o modelo relacional e o conceito de *time sequences* (TS's) de Segev (seção 2.1.9). Cada TS pertence a uma relação TS, possui uma chave única e serve de unidade de manipulação e recuperação de dados. O modelo suporta tempo de transação e tempo válido armazenados como pontos de tempo. Como descritas originalmente, as TS's são capazes de representar históricos do tipo contínuo, discreto e escada.

Exemplo:

Poderíamos representar a relação R neste modelo na forma:

VALID	*NOME*	*DEPTO*	*TIMESTAMP*
1	Luís	Vendas	t1
4	Luís	Nulo	t2
5	Luís	Pessoal	t3
9	Luís	Nulo	t4
2	Rita	Pessoal	t5
7	Rita	Nulo	t6

Mais uma vez utilizamos valores nulos para representar entidades não existentes em certos períodos de tempo. Os valores t1 a t6 representam os tempos de criação das respectivas tuplas.

□

Com o suporte aos dois tipos de tempo o modelo adquire a capacidade de realizar mudanças retroativas e pré-ativas. As principais operações do modelo, expressas numa linguagem baseada no SQL, são:

T_CREATE TS: insere uma TS numa relação TS;

T_UPDATE TS: atualiza uma TS criando uma nova tupla;

T_CORRECT TS: altera o valor de uma tupla de uma TS, contudo a tupla antiga não é perdida;

T_DELETE TS: uma "lápide" é criada, não permitindo mais atualizações sobre a TS;

T_REMOVE TS: remove fisicamente os dados de uma TS;

T_SELECT TS: recupera TS's (com todas as suas tuplas ou com parte delas).

Nas quatro primeiras operações o tempo válido deve ser fornecido pelo usuário. Em todas as operações é necessária a especificação das relações TS, por meio da cláusula T_FROM, e das TS's em si, pela cláusula T_WHERE. A cláusula T_WHERE realiza uma seleção verificando-se a validade de um predicado em algum ou em todos os pontos de tempo de um conjunto de instantes de tempo. Este conjunto é determinado pela indicação explícita de um instante ou intervalo ou por uma condição sobre os dados. Dadas as TS's que satisfaçam à cláusula T_WHERE, a operação T_SELECT permite especificar se serão recuperadas todas as suas tuplas, apenas as tuplas que satisfaçam à cláusula, ou somente as tuplas válidas em um instante ou intervalo de tempo específico.

Os autores argumentam que o modelo descrito deveria ser mapeado para um modelo de dados de objetos complexos, onde o histórico de um objeto pudesse ser modelado como um objeto complexo. As principais vantagens advindas desta estratégia seriam a facilidade de implementação, a eficiência na manipulação de dados temporais, a flexibilidade de criação de caminhos de acesso e o tratamento uniforme de dados temporais e não temporais.

A partir desta decisão os autores mapeiam o modelo temporal proposto para um modelo de objetos complexos denominado modelo MAD (*molecule-atom data model*), e a linguagem temporal para a linguagem de consulta do modelo (MQL). O sistema cria e manipula automaticamente atributos para identificação única, tempo de transação e indicação de exclusão de uma TS. O usuário é responsável pela criação de um atributo explícito para tempo válido (e pela determinação

de sua granularidade). Uma característica do modelo de implementação é a manutenção da versão mais jovem de cada TS armazenada totalmente enquanto as versões anteriores contêm nulos nos atributos que não sofrem alteração. Outra política adotada é a separação de dados históricos da versão mais adiantada no tempo (nem sempre a corrente).

Em [KS92] os autores propõem a extensão do próprio modelo MAD para um modelo temporal (*temporally oriented MAD — TMAD*) e descrevem como maneira de implementá-lo um mapeamento para o modelo original MAD. O modelo TMAD será referenciado como Käfer_2 na análise comparativa dos diversos modelos por se basear em um modelo de objetos complexos, diferenciando-se do primeiro modelo (referenciado como Käfer) baseado no relacional.

No modelo MAD cada instância dos tipos do esquema é um átomo e uma molécula consiste no conjunto de átomos ligados da forma estabelecida na cláusula FROM de uma consulta. No modelo estendido objetos passam a ser históricos de moléculas (um histórico de molécula descreve os estados de uma molécula em um intervalo de validade).

As consultas são feitas de modo similar ao modelo relacional temporal, por meio de T_SELECT, T_FROM e T_WHERE. T_WHERE especifica um predicado para seleção de moléculas e tempo para verificação deste predicado. T_FROM define a construção da molécula, enquanto T_SELECT define em que tempo serão recuperados os dados das moléculas selecionadas. Atualizações, feitas pela cláusula T_UPDATE, ocorrem sem remoção de estados anteriores. O tempo de transação é suprido automaticamente pelo SGBD, enquanto o tempo válido deve ser fornecido pelo usuário. Os autores não mencionam as operações de inserção, remoção ou alteração de dados passados.

No mapeamento de TMAD para MAD os seguintes atributos são incluídos na definição de cada tipo (*atom type*): VALID_FROM, VALID_UNTIL (tempo válido), TRANSACTION_TIME (tempo de transação), PAST e FUTURE (apontam para o estado anterior e posterior do átomo).

Exemplo:

A relação R poderia ser representada no novo modelo como:

OID	*NOME*	*DEPTO*	*VALID_FROM*	*VALID_TO*	*T_TIME*	*PAST*	*FUTURE*
#1	Luís	Vendas	1	3	t1	Nulo	p1
#1	Luís	Pessoal	5	8	t2	p2	Nulo
#2	Rita	Pessoal	2	6	t3	Nulo	Nulo

Os tempos t1, t2 e t3 representam os instantes de criação de cada tupla. O ponteiro p1 aponta para a segunda tupla e p2 aponta para a primeira.

□

Finalmente, os autores descrevem um mapeamento das consultas e atualizações temporais de TMAD para o modelo MAD.

2.1.15 Gabbay

Para Gabbay e McBrien [GM91], um banco de dados histórico (D_T) é considerado uma série de BD's relacionais D_t's, onde t corresponde ao tempo associado a cada BD particular. Todos os D_t's

possuem o mesmo esquema — considera-se que D_T inclui todas as relações que existiram sobre o período de vida do BD histórico. Então, se o esquema de uma relação pertencente a D_T é alterado, os autores consideram que ambas as relações (com esquema antigo e novo) estarão incluídas em D_T. Cabe ao usuário decidir o que acontecerá aos dados existentes no momento da alteração.

A álgebra proposta para o modelo é baseada na álgebra relacional e na lógica US (derivada da lógica de primeira ordem com a inclusão dos operadores SINCE e UNTIL). Dada uma consulta sobre um momento q (BD D_q) e duas fórmulas da lógica A e B, A SINCE B será verdadeiro se: B for verdade para um D_s anterior a D_q e A for verdade para todo D_j desde (inclusive) D_s até (exclusive) D_q. Similarmente A UNTIL B será verdadeiro se: B for verdade para um D_u posterior a D_q e A for verdade para todo D_j desde (exclusive) D_q até (inclusive) D_u.

Os cinco operadores da álgebra relacional ($\cup$, $-$, $\times$, σ e π) permanecem inalterados, sendo utilizados para a manipulação de um D_t específico. Dois novos operadores, SINCE-PRODUCT (S_X) e UNTIL-PRODUCT (U_X), são incluídos para projetar para um D_t específico tuplas de BD's relativos a outros instantes de tempo. A operação A S_X B emparelha todas tuplas x e y, tal que, x foi membro da relação A em todos os D_t's anteriores ao corrente, a partir do instante em que y foi membro da relação B. A operação A U_X B é definida de maneira semelhante. Com base nestes dois operadores, seis novos operadores são definidos, servindo para selecionar dados presentes em algum D_t posterior a q, em todos os D_t's posteriores a q ou no D_t imediatamente posterior q (e seus equivalentes com relação a D_t's anteriores a q), onde q representa o tempo especificado para a consulta.

No mesmo artigo os autores descrevem mecanismos para transportar seu modelo relacional temporal para o relacional tradicional. Cada tabela do BD é aumentada com a inclusão de dois atributos (*START_TIME* e *END_TIME*), ou seja, os autores escolhem efetuar a marcação de tempo a nível de tupla, com intervalos. Também é fornecido um mapeamento dos operadores da álgebra relacional temporal para expressões da álgebra relacional tradicional.

Exemplo:

Continuando com nosso exemplo, neste modelo a relação R seria representada como:

NOME	*DEPTO*	*START_TIME*	*END_TIME*
Luís	Vendas	1	3
Luís	Pessoal	5	8
Rita	Pessoal	2	6

□

Finalmente, os autores apresentam Temporal-SQL, uma extensão do SQL que suporta os novos operadores temporais. A cláusula FROM é estendida para permitir as opções SINCE e UNTIL, correspondendo aos operadores SINCE-PRODUCT e UNTIL-PRODUCT. Toda consulta deve ser prefixada com a cláusula AT, que indica o tempo (D_t) a ser utilizado como base na consulta. Os operadores temporais derivados de SINCE-PRODUCT e UNTIL-PRODUCT também são permitidos através das cláusulas FUTURE, ALWAYS-WILL, NEXT, PAST, ALWWAYS-HAS e PREVIOUS. Os autores apresentam ainda, uma estratégia para a tradução de Temporal-SQL para SQL.

2.1.16 Jensen

Jensen, Mark e Roussopoulos consideram apenas a dimensão de tempo de transação em sua proposta [JMR91]. Outras características do modelo são a manutenção da primeira forma normal, ou seja, associação de tempos às tuplas, e política de acesso eficiente também aos dados históricos, em contraste com a estratégia de priorizar o acesso aos dados correntes.

A opção por tempo de transação tem como conseqüência o suporte a BDTT, no qual dados passados não podem ser alterados ou removidos. Entretanto, os autores citam a possibilidade de remoção de dados históricos por motivos administrativos. Além disto, os tempos para a marcação das tuplas são fornecidos automaticamente pelo sistema. O modelo adota ainda marcação de tuplas com pontos de tempo e variação de valores conforme uma função escada.

Para a conservação de dados históricos são utilizados *backlogs*. Um *backlog* B_R para uma tabela R é uma relação que contém o histórico completo das requisições de modificação efetuadas contra a relação R. Cada tupla de um *backlog* corresponde a uma operação (inserção, exclusão ou atualização) executada sobre uma tupla de R. *Backlogs* possuem os mesmos atributos de suas relações originais e três atributos adicionais. O atributo *ID* corresponde ao *surrogate* da tupla que sofre a modificação; tal *surrogate* é gerado pelo sistema e não é visível nem alterável pelo usuário. O atributo *OP* indica a operação que a tupla representa — inserção, exclusão ou atualização. Por fim, *TTIME* representa o tempo de transação, correspondente ao instante em que a transação que executa a modificação emite seu *commit*. Observe-se que as tuplas de um *backlog* estão ordenadas pelo tempo de transação, na seqüência em que foram inseridas.

Exemplo:

Se considerarmos o tempo de transação igual ao tempo válido, uma possível representação do *backlog* da relação R seria:

ID	*OP*	*TTIME*	*NOME*	*DEPTO*
#E1	INS	1	Luís	Vendas
#E2	INS	2	Rita	Pessoal
#E1	DEL	4	Luís	Vendas
#E1	INS	5	Luís	Pessoal
#E2	DEL	7	Rita	Vendas
#E1	DEL	9	Luís	Pessoal

□

Para recuperar uma relação base é necessário primeiramente definir o tempo a que ela se refere. O instantâneo da relação assim encontrado (denominado fatia da relação) pode ser computado sempre que necessário ou armazenado segundo uma dentre duas maneiras: armazenamento de seus dados como uma relação de fato (materialização) ou utilização de uma matriz de ponteiros para as tuplas do *backlog* que correspondem à relação no tempo procurado (*cache* de índice). As fatias das relações base podem ser fixas, fatias referentes a um tempo fixo, ou dependentes de tempo, onde o tempo das fatias é especificado através de uma expressão que envolve o tempo corrente. Os

conceitos acima aplicados a relações, são também válidos para visões, com alguns novos detalhes (e.g., visões sobre relações materializadas podem referenciar diretamente esta relações, ao invés de seus *backlogs*). Visões derivadas diretamente de *backlogs* são denominadas visões de *backlogs*.

Fatias de relações base, visões e visões de backlogs armazenadas e dependentes de tempo tornam-se desatualizadas com o passar do tempo. O conjunto de modificações necessárias para atualizar tais tabelas é denominado arquivo diferencial. Arquivos diferenciais de relações base e visões de *backlog* são armazenados fisicamente como parte do *backlog*; para outros tipos de visões, arquivos diferenciais são apenas conceituais.

A eficiência de acesso aos dados pode ser aumentada através do armazenamento de fatias de relações/visões causando, contudo, um crescimento na redundância dos dados. Novas fatias das relações/visões podem ser computadas incremental ou decrementalmente de outras fatias já armazenadas. Relações/visões dependentes de tempo podem ser atualizadas através de uma política de *eager* ou *lazy evaluation*. Todas estas variáveis devem se controladas dinamicamente para se obter a melhor razão eficiência de acesso/redundância de dados para o sistema.

Os autores estendem ainda a álgebra relacional para lidar com o modelo. Os operadores básicos (σ, π, $\cup$, $-$ e $\times$) não são alterados. Para manipular a dimensão temporal é incluído o operador TIME SLICE, equivalente a restringir uma relação/visão a um instante de tempo específico. Complementarmente pode-se acessar diretamente os *backlogs* das relações base, fazendo-se restrições e projeções sobre seus atributos (inclusive *TTIME* e *OP*).

2.1.17 Su

Su e Chen [SC91] estendem um modelo de dados denominado OSAM e sua linguagem de consulta OQL para modelar e acessar um BDT. Adota-se *timestamping* de objetos por intervalos, acrescentando-se os atributos *START_TIME* e *END_TIME*. O modelo suporta apenas uma dimensão temporal, contudo os autores afirmam que novas noções de tempo podem ser incorporadas ao modelo através da utilização de regras. Infelizmente, o uso destas regras não é detalhado no artigo, deixando obscuro o poder de tal abordagem.

São permitidas correções a dados válidos no passado ao administrador do BD. Contudo, estas correções ocasionam a remoção total dos dados alterados — não deixam quaisquer sinais dos erros corrigidos. Alterações retroativas sem perda de dados seriam possíveis com o uso de regras para incorporar outras dimensões de tempo.

A versão corrente dos objetos mantém seus dados por completo, enquanto as versões históricas mantém apenas os dados referentes às alterações ocorridas durante a evolução do objeto. As versões históricas são ainda separadas lógica e fisicamente das correntes.

No modelo OSAM um objeto pode estar presente em várias classes. Uma instância é a representação de um objeto numa classe específica e é unicamente identificada pela identificação do objeto e da classe. Os *timestamps* são associados às instâncias de objetos, o que resulta na manutenção de históricos de instâncias.

A principal extensão à linguagem de consulta é a introdução da cláusula (opcional) WHEN para especificar o período de tempo de interesse para a consulta. Este período pode ser indicado diretamente por um intervalo de tempo ou através de uma condição sobre os dados. Neste caso, o período é composto pelos intervalos de tempo em que a condição é satisfeita. São acrescentados à linguagem um conjunto de funções temporais, operadores de comparação de intervalos e operadores de conjunto para lidar com os dados temporais. Funções temporais incluem tarefas tais como

encontrar instâncias anteriores/posteriores a um momento de tempo ou a uma instância específica e obter o tempo inicial/final de validade de um determinada instância. Operações de comparação de intervalos englobam comparações como anterior, posterior, adjacente ao início/final do intervalo, contido no início/final do intervalo e outras. Dois operadores (*ANY* e *EVERY*) são introduzidos para expressar condições periódicas num intervalo de tempo, correspondendo às noções de $\exists$ e $\forall$. Operadores de conjunto (união, interseção e diferença) são adicionados para possibilitar a junção dos resultados de duas ou mais consultas.

2.1.18 Wuu

Wuu e Dayal apresentam em [WD92] uma proposta de extensão do modelo funcional orientado a objetos OODAPLEX. Neste modelo todas as propriedades de objetos, relacionamentos entre objetos e operações sobre os objetos são modeladas por funções aplicadas sobre os objetos. Na extensão temporal estas funções são alteradas para retornar uma outra função que mapeia valores de tempo para valores instantâneos. Por exemplo, uma propriedade *DEPTO* deixaria de ser uma função que retorna o nome do departamento para cada empregado, passando a retornar uma outra função que, por sua vez, mapeia instantes de tempo para os nomes de departamento.

Várias dimensões de tempo podem ser suportadas utilizando-se esta abordagem — cada dimensão seria modelada como um argumento nas funções de mapeamento entre tempos e valores. O tempo de transação é mantido automaticamente pelo SGBD, enquanto os valores referentes a tempo válido devem ser supridos pelo usuário. A modelagem por funções permite ainda a marcação de tempo a nível de objetos ou atributos. Para representação dos *timestamps* os autores optam por intervalos de tempo.

Cada objeto tem associada uma função *lifespan*, que retorna o intervalo em que o objeto existiu no BD. Embora sejam permitidas migrações entre classes, o *lifespan* total de cada objeto deve ser contíguo.

A linguagem de consulta do modelo OODAPLEX não é alterada para o novo modelo. Consultas temporais são executadas no modelo pela possibilidade de acesso direto aos valores de tempo.

2.1.19 Outros Artigos

Além das propostas descritas acima, existem diversos trabalhos que também tratam de aspectos de BDT's. Nesta seção exporemos resumidamente alguns trabalhos que não apresentam um modelo completo de dados ou que enfocam aspectos de implementação.

Clifford e Croker [CC88] apresentam alguns aspectos que devem ser capturados por SBD's orientados a objetos que pretendam lidar com tempo. Um objeto histórico é definido como um objeto cujos atributos denotam funções, que definem um mapeamento de objetos do tipo tempo para objetos do tipo especificado pelo atributo. Cada objeto histórico possui um período de validade que consiste no conjunto de períodos de tempo em que o objeto esteve "ativo" no BD. Os autores exibem então primitivas para criação, exclusão, "ressurreição", remoção definitiva e junção de objetos.

Date expõe em [Dat88] uma estratégia para adicionar o tipo de dados tempo ao SQL. São definidas várias características do tempo a ser suportado — granularidade, origem, escopo, representação por intervalos e datas — e as operações possíveis sobre dados de tempo. O tratamento dispensado neste artigo não engloba características de BDT's, como o armazenamento de dados históricos.

Em [Cha88], o autor considera a questão de tempo relativo (relacionamentos do tipo antes, depois e durante). O artigo trata de relacionamentos binários apenas, e não considera a variação de valores no tempo, como a ocorrida em dados históricos. O foco da proposta está no tratamento de atributos de tempo, como por exemplo, data de nascimento, de contratação, de promoção, etc. Chaudhuri propõe um modelo de grafo em que os nós representam atributos de tempo e valores temporais absolutos, enquanto as arestas representam relacionamentos temporais. O BD é mapeado para o grafo, e este então pode ser utilizado para expressar relacionamentos genéricos entre os atributos temporais, como também para prover uma estrutura conceitual para auxiliar no processamento de consultas temporais expressadas em tempo relativo.

Em [KYL90] é descrita uma linguagem de consulta a BDT's, chamada ETQL (*Extend Temporal Query Language*). A linguagem é uma extensão de TQuel, de Snodgrass [Sno87], que visa fornecer suporte ao que os autores denominam de tempo abstrato ou relativo. Este tipo de tempo engloba expressões do tipo "primavera de 1988", "segunda versão", "dia de fundação da empresa", "Natal do ano passado", etc. EQTL difere ainda de TQuel por não suportar tempo de transação e pela introdução de alguns novos operadores para comparação temporal. A linguagem foi implementada como um *front end* do SGBD Ingres.

Em [EWK90] os autores introduzem uma técnica de indexação para melhorar o desempenho de operações típicas de BDT's. A técnica se constitui na escolha de um conjunto de pontos de tempo para indexação e no uso de B^+ *trees*. Os índices assim formados, denominados *time indexes*, diferem da simples aplicação de B^+ *trees* porque são baseados em objetos cuja pesquisa é efetuada por intervalos.

Leung e Muntz [LM90] também consideram a questão de processamento/otimização de consultas para BDT's. Os autores usam como base o modelo de Segev (seção 2.1.9) para introduzir um mecanismo que aproveita a ordem dos dados para o processamento de consultas. É citada ainda a oportunidade de otimização baseada na semântica dos dados temporais.

Maiocchi e Pernici [MP91] realizam uma análise comparativa de sistemas gerenciadores de dados temporais divididos em diversas áreas, tais como, bancos de dados, inteligência artificial e engenharia de software. São consideradas para comparação questões como raciocínio temporal, tempo relativo, manipulação de informações incompletas e dimensões temporais. Das propostas analisadas, a única relativa a BD's é TQuel de Snodgrass [Sno87].

Croker e Clifford [CC89] definem um cálculo de tupla denominado *Lh* que deveria servir como base de comparação para linguagens de consulta históricas. Este cálculo é definido com respeito a uma estrutura de relações históricas, denominada *Canonical Historical Model*, que possui, segundo os autores, capacidade de modelagem pelo menos igual às propostas apresentadas até a publicação do artigo. Tuzhilin e Clifford apresentam em [TC90] um complemento à proposta de Croker e Clifford, com a introdução de mais um cálculo de tupla e uma álgebra com o mesmo propósito de *Lh*. As propostas possuem o mesmo poder de expressividade, são reduzidas ao cálculo/álgebra relacional quando o tempo consiste de um único instante e são independentes de qualquer modelo temporal relacional específico. Estas características, argumentam os autores, os fazem adequados para o estabelecimento da noção de completeza relacional histórica. Entretanto, uma análise a este nível foge ao escopo desta revisão.

McKenzie e Snodgrass revisam em [MS91] doze álgebras relacionais estendidas para incorporar a dimensão temporal. Inicialmente os autores descrevem algumas características de BDT's, passando a seguir a descrever as diversas álgebras. As álgebras são avaliadas com relação a 26 critérios, alguns destes introduzidos pelos autores e o restante oriundos de diversos artigos na área

de BDT's. Existem dois critérios cuja satisfação, individual ou conjunta, implica na impossibilidade de satisfação de outros critérios — ao todo sete critérios são conflitantes. Os autores afirmam que os 26 critérios propostos para a avaliação são independentes, isto é, considerando-se os critérios dois a dois não há critérios equivalentes ou incompatíveis (exceto os critérios conflitantes), e podem servir como base para avaliar novas propostas de álgebras temporais.

Jensen e Snodgrass introduzem em [JS92] uma taxonomia para relações temporais, baseados nos conceitos de tempo válido e tempo de transação do modelo de Snodgrass. Apesar destes dois tipos de tempo serem independentes, em algumas aplicações eles mantêm algum tipo de relacionamento; a taxonomia é proposta a partir destes relacionamentos. A classificação leva em consideração aspectos como a limitação dos valores de tempo válido a partir dos valores do tempo de transação (e.g., uma relação onde o valor de tempo válido de um elemento, tupla ou atributo, é sempre menor que o tempo de transação), relacionamentos entre *timestamps* de elementos da relação (e.g., sempre que uma informação é armazenada após outra, terá validade somente após ela) e regularidade dos valores dos *timestamps* (e.g., a cada três dias é gravada uma nova informação).

Snodgrass apresenta em [Sno92] uma proposta para a criação de uma linguagem de consulta consensual para BDT's relacionais (*Temporal Structured Query Language — TSQL*). Tal projeto seria uma tentativa de unir esforços para a formação de uma base sólida para futuras pesquisas em BDT's. Para tanto, o autor relaciona um conjunto de tarefas necessárias à realização do projeto e estabelece uma ordem parcial entre estas, baseada nos pré-requisitos de cada tarefa. Em [JCG+92] temos um primeiro artigo nesta direção, visando o estabelecimento de uma terminologia comum no campo de BDT's. Esta terminologia é adotada nesta dissertação.

2.2 Análise Comparativa das Propostas

Nesta seção fazemos uma análise comparativa dos diversos trabalhos descritos na seção anterior, comparando-os em relação a uma série de tópicos que consideramos importantes para o entendimento do problema. Ao final da seção sumarizamos as comparações através de quadros sinópticos.

2.2.1 Modelo de Dados Base

A grande maioria dos trabalhos aqui apresentados baseia-se no modelo de dados relacional. As exceções ficam por conta de dois modelos (Adiba e Klopporgge) baseados no modelo ER, um modelo concebido independentemente (Segev), um modelo baseado em um modelo de dados de objetos complexos (Käfer$_2$) e duas propostas (Su e Wuu) que estendem o modelo OO.

Um grande número de trabalhos estendendo o modelo relacional para incluir tempo foi produzido na última década. Agora, há uma tentativa de concentrar estes esforços para se alcançar um consenso [Sno92, JCG+92]. Por outro lado, as propostas de BDTOO's começaram a aparecer apenas recentemente e são ainda em pequena quantidade. A proposta de Su não considera em profundidade as implicações da introdução de questões temporais em algumas características básicas do paradigma OO, como herança. O modelo de Wuu é baseado em um modelo de dados OO funcional, e deixa ao usuário do BD a definição de grande parte do modelo de dados temporal. Nós concordamos com Käfer e Schöning [KS92] em considerar que as definições de um modelo de dados temporal são demasiado complexas para serem deixadas ao usuário e deveriam ser providas pelo SGBD.

2.2.2 Dimensões Temporais

A questão de dimensões temporais é, indubitavelmente, um aspecto fundamental em um modelo temporal. O conceito de dimensões temporais, bem como a classificação dos tipos de tempo, são apresentados na seção 1.3.1.

No modelo de Snodgrass provê-se suporte a uma dimensão de tempo válido e uma de transação. O modelo inicial de Gadia é restrito a uma dimensão temporal (tempo válido), porém , na sua extensão final [GY88], o modelo é generalizado para o suporte a um número arbitrário de dimensões temporais.

Lum afirma a necessidade da manutenção de dois tipos de tempo. Apesar disto, o seu sistema mantém apenas o tempo físico (de transação), deixando ao usuário o tratamento de uma ou mais dimensões de tempo lógico (válido).

Sarda reconhece a existência de duas dimensões de tempo: tempo de transação e tempo de mundo real (válido). O seu modelo está voltado ao suporte deste último, contudo pode suportar tempo de transação, se o usuário assim o solicitar.

Navathe adota estratégia semelhante a Sarda. O tempo válido é suportado por todas relações variantes no tempo, e outros tipos de tempo, como tempo de transação, podem ser acrescentadas se solicitado pelo usuário. Porém, não são definidas operações sobre a dimensão do tempo de transação.

O modelo de cubos de dados de Ariav suporta um número arbitrário de dimensões temporais através da definição de TSA's (*timestamp attributes*). O sistema define e mantém automaticamente um TSA especial para tempo de transação.

O modelo de Su provê suporte a apenas uma dimensão temporal. Seus autores afirmam no entanto, que outras dimensões de tempo podem ser acrescentadas ao modelo através do uso de regras. A abordagem entretanto não é detalhada no artigo de descrição do modelo [SC91].

Nos modelos de Käfer e Käfer_2, são mantidos tempo válido e de transação. Contudo, assim como em Navathe, não são definidas operações que atuem sobre a dimensão do tempo de transação.

O modelo funcional de Wuu possibilita a manutenção de um número arbitrário de dimensões de tempo. Cada dimensão temporal é implementada como mais um argumento para as funções do modelo.

O modelo de Abbod prove suporte a tempo válido e de transação. Adiba, Klopprogge e Jensen restringem-se ao suporte a tempo de transação. Clifford, Segev, Tansel, Lorentzos e Gabbay consideram apenas uma dimensão temporal, relacionada a tempo válido.

2.2.3 Variação de Valores no Tempo

Entidades do BD (atributos, tuplas, componentes, objetos) podem variar no decorrer do tempo de várias maneiras. Os diversos tipos de variação de valores existentes na literatura são descritos na seção 1.3.2.

A variação escada é suportada por *default* em todos os modelos aqui descritos, excetuando-se Klopprogge. Neste modelo, associa-se a cada histórico uma função de derivação e um conjunto de funções de aproximação, definidas pelo usuário, para a determinação de estados não armazenados. Os modelos de Segev e Käfer suportam os quatro tipos de variação de valores, determinadas pelo tipo das TS's. Além da variação escada os modelos de Sarda e Snodgrass suportam a variação discreta, através da definição de relações evento e intervalo. Embora Clifford cite a existência de outras funções de interpolação, seu modelo se restringe a função escada. Finalmente, o modelo de

Lorentzos suporta a variação discreta com relações que representam o tempo por pontos e utiliza o operador COMPUTE para simular outros tipos de interpolação.

2.2.4 Evolução de Esquema

A manutenção de dados históricos traz à tona a questão de evolução de esquema. O esquema de um BDT pode, naturalmente, sofrer várias reestruturações durante o decorrer do tempo e, por conseguinte, seus dados estarão associados a diversos esquemas. A recuperação destes dados passa então a ser mais complexa. Apesar da importância deste tópico, ele é totalmente ignorado ou visto superficialmente na maioria dos modelos aqui analisados — apenas dois artigos [MNA87, MS90] tratam especificamente desta questão.

Lum e Adiba adotam uma solução semelhante para o problema: o esquema recebe o mesmo tratamento que as relações do BD; deste modo, todos os esquemas de um BDT são armazenados e podem ser utilizados na recuperação de dados históricos. No modelo de Gabbay, todos os esquemas de cada relação do BDT são considerados como parte do esquema do BDT, possibilitando a recuperação de dados dos diversos esquemas. Entretanto, estas propostas não dão atenção ao tema, não descrevendo detalhes de como lidar com estes diversos esquemas.

No modelo de Clifford cada atributo de uma relação tem associado a si os períodos de tempo em que esteve "ativo" no esquema da relação. Através desta técnica, Clifford afirma que seu modelo está apto a suportar esquemas variantes no tempo. Todavia esta abordagem possui algumas limitações, como a exigência de que atributos chaves sejam imutáveis e válidos por toda a vida da relação.

Snodgrass [MS90] formaliza o conceito de evolução de BD's através da definição de uma linguagem algébrica que oferece suporte a evolução e versionamento de esquemas e dados. O BD é visto como uma seqüência de estados, nos quais cada relação está ligada a um esquema, que pode ser diferente em cada um dos estados. O foco da abordagem é sobre tempo de transação, tendo em vista que a questão de evolução de esquema é relacionada a este tipo de tempo. Todavia, a linguagem algébrica engloba os dois tipos de tempo (válido e de transação) e os quatro tipos de relações deles resultantes (instantâneas, de tempo válido, de tempo de transação e bitemporais).

Navathe [MNA87] objetiva primariamente propor um mecanismo de atualização não destrutiva de esquema que facilite a consulta a dados de esquemas antigos. A simples manutenção de esquemas históricos não é suficiente, pois obriga o usuário a conhecer todos os esquemas passados do BD para a execução de algumas consultas. É proposta então uma lógica temporal de esquema (STL) para lidar com mudanças de esquema. Um conjunto de fórmulas é associado a cada esquema, para descrever sua estrutura e evolução; estas fórmulas são utilizadas para traduzir consultas temporais para os diversos esquemas.

2.2.5 Nível de Registro de Tempo

Este item diz respeito à escolha de associar *timestamps* às entidades ou seus componentes (no modelo relacional, a tuplas ou atributos). A opção pelo nível de tuplas é geralmente justificada pela manutenção da primeira forma normal, pela fácil manipulação e possibilidade de implementação mais rápida (geralmente alterando-se um SGBD convencional). A escolha por atributos diminui a redundância nos dados temporais, tendo em vista que os eventos no BD nem sempre alteram todos os atributos numa tupla. Por fim, a marcação do tempo a nível de atributos evita o que Gadia em [Gad88] chama de anomalia temporal vertical, isto é, a dispersão das informações sobre uma entidade em diversas tuplas.

Os modelos de Snodgrass, Segev, Sarda, Ariav, Lorentzos, Lum, Abbod, Käfer, Navathe, Jensen e Gabbay efetuam a marcação de tempo a nível de tupla. Optam pelo nível de atributo os modelos de Clifford, Tansel e Gadia. O modelo de Klopprogge, baseado no ER, registra o tempo a nível de componentes (atributos e *roles*). Adiba deixa ao usuário a escolha do nível de registro de tempo, dado que um tipo pode ser constituído de vários níveis de construção. No modelo de Wuu, o usuário deve escolher o nível de registro de tempo. Por fim, nos modelos de Su e Käfer_2, *timestamps* são associados aos objetos.

Além dos valores de tempo, alguns modelos associam aos objetos um indicador do período de sua existência. Uma das características das TS's de Segev, a saber, o seu *life span*, descreve o intervalo de validade de cada TS. Tansel propõe a inclusão, em todas as relações, de um atributo para expressar o período de validade da tupla. No modelo de Klopprogge a existência dos objetos é indicada por meio de um atributo obrigatório associado a cada entidade. No trabalho de Adiba, a existência de um objeto é determinada pelo que denomina de sua persistência, que indica quantas das suas últimas versões devem ser consideradas.

As estratégias que Clifford adota para indicar os períodos de tempo em que as entidades são retratadas pelo BD sofrem alterações ao longo do tempo. Em seu artigo inicial [CW83] é proposta a inclusão de um atributo booleano obrigatório para indicar a existência ou não de cada entidade em um estado do BD. Em [CT85] a existência das entidades é determinada por um *lifespan* associado a cada relação e pelo uso de valores nulos. No terceiro artigo do modelo [CC87] a estratégia é novamente alterada — o *lifespan* outrora associado às relações passa a estar relacionado com os atributos e o uso de nulos é substituído pela marcação de cada tupla com um *lifespan*.

2.2.6 Representação do Tempo

A representação de dados temporais nos modelos tradicionais se faz pela associação de marcas de tempo aos seus componentes (tuplas ou atributos). Estes registros de tempo podem ser de dois tipos básicos: pontos e intervalos. A representação por intervalos tem a vantagem de que cada tupla (atributo) é autocontida (o), ou seja, possui a informação do tempo de início e de fim da validade de um estado qualquer. Na representação por pontos é necessário encontrar a próxima tupla (ou o valor do atributo na tupla) da entidade para se descobrir o tempo final de validade do valor anterior. Todavia, a representação por intervalos é adequada somente para dados que variem conforme uma função degrau. Dados que representem eventos ou funções contínuas são melhor representados por pontos de tempo.

Snodgrass e Lorentzos permitem as duas formas de representação em seus modelos. Segev, Ariav, Abbod, Käfer, Adiba, Klopprogge e Jensen escolhem a representação por pontos, enquanto que Navathe, Sarda, Gabbay, Wuu, Käfer_2 e Su optam por intervalos. No modelo de Gadia os valores dos atributos são associados a um conjunto de intervalos e não a um intervalo. Tansel opta inicialmente pela representação por intervalos. No entanto, na sua extensão para o modelo relacional aninhado, a representação é trocada pela escolhida por Gadia. No modelo de Clifford, o valor de um atributo corresponde a um mapeamento de pontos de tempo para valores de dados. Em contrapartida os *lifespans* dos atributos das relações e das tuplas correspondem a conjuntos de intervalos. Por fim, Lum afirma a necessidade da representação por intervalos na descrição de seu modelo base. Contudo, ao descrever as estruturas para implementação do modelo, utiliza-se da representação por pontos.

2.2.7 Implementação

Vários dos trabalhos aqui apresentados foram implementados integral ou parcialmente, geralmente alterando-se SGBD's já existentes para acrescentar o suporte temporal. Os protótipos assim surgidos têm como principal objetivo demonstrar a factibilidade de seus respectivos modelos e, normalmente, não levam em consideração questões de eficiência. As informações abaixo referem-se aos estados das implementações nas datas de publicação dos artigos de descrição do modelo.

O modelo de Snodgrass foi implementado através da alteração do SGBD Ingres, tendo sido utilizado inclusive para a execução de um *benchmark* com operações temporais [AS86]. Conceitos de evolução de esquema não foram incluídos no protótipo por terem sido formalizados no modelo posteriormente. Lorentzos e Abbod também utilizam-se do Ingres para a implementação de seus modelos. Lorentzos menciona a existência de um protótipo de seu modelo, implementado com sucesso. Abbod cita a existência de duas implementações de seu modelo (SISBASE I e SISBASE II), a primeira não engloba todos os conceitos do modelo enquanto a segunda é referenciada como estando em desenvolvimento.

Sarda menciona a implementação de um SGBD histórico experimental de seu modelo; Klopprogge cita a existência de um compilador que executa o mapeamento de uma interface de seu modelo temporal para um SGBD do modelo de rede. Adiba, Arlav e Lum referem-se a implementações em andamento de seus modelos. Käfer, Käfer_2, Wuu e Su descrevem com detalhes o mapeamento de seus modelos para SGBD's já existentes. Jensen expõe um modelo de implementação de seu modelo, porém não fica claro se tais implementações foram de fato realizadas. Similarmente, Tansel expõe uma linguagem gráfica de consultas (TBE), cuja implementação não é citada. No artigo inicial do modelo de Segev [SK86] estruturas físicas para a implementação do modelo são descritas, no entanto não há referência à sua real implementação. Clifford, Gadia, Navathe e Gabbay não mencionam a existência de protótipos de seus modelos.

2.2.8 Tratamento de Alterações

Como dissemos na seção 1.3.1 a origem do conceito de BDT's está ligada fortemente às idéias de manutenção de estados passados do BD e de indicação do tempo de validade no mundo real dos dados do BD. Nesta seção analisaremos a questão de alterações aos dados do BD, tendo em vista dois tópicos diretamente relacionados a estas questões, a saber, a preservação de informações (capacidade de se recuperar qualquer dado que tenha passado pelo BD) e alterações retroativas (capacidade de se modificar dados cuja validade no mundo real está associada a tempos do passado).

No modelo de Snodgrass, as alterações dependem do tipo de relação que está sendo considerada. Em relações de tempo de transação, alterações retroativas não são possíveis; relações de tempo válido permitem-nas, porém com a perda do conteúdo anterior. Em relações bitemporais, todas as alterações ao BD geram um novo estado histórico, o que possibilita modificações retroativas sem perda dos valores antigos. O modelo de Jensen objetiva o suporte de um BDTT e, portanto, exibe o mesmo comportamento das relações de tempo de transação de Snodgrass. Jensen porém menciona a possibilidade de remoção física de dados por motivos administrativos.

No modelo de Gadia, são permitidas alterações retroativas, porém o conteúdo antigo é perdido. Sarda também permite alterações retroativas, sendo que, para evitar a perda dos dados antigos o usuário deve requerer o suporte a tempo de transação. Abbod utiliza um esquema de versionamento para permitir modificações retroativas, sem perda de dados. Käfer possibilita alterações às informações históricas mantendo as informações antigas. O modelo fornece ainda a opção de

remoção física de dados que não interessem mais.

Atualizações ao estado corrente, no modelo de Adiba, podem causar a perda da versão anteriormente corrente, no caso de históricos periódicos e manuais. O número de versões históricas mantidas pode ser limitado, o que leva a descartar as versões mais antigas de um histórico. Além disto, a exclusão de um objeto não histórico, mas que contenha atributos históricos, causa a remoção física de todos os seus dados. O comando de correção permite alterar dados válidos no passado, pela substituição por novos dados, ou seja, perdendo informações.

Lorentzos define operadores algébricos que permitem manipular dados temporais, inclusive realizar alterações retroativas. Não é definido entretanto nenhum mapeamento entre operações de atualização e os operadores algébricos. A utilização de apenas uma dimensão temporal no entanto, faz com que dados passados sejam perdidos após alterações.

O modelo de Lum permite alterações retroativas. Todavia, o tratamento de modificações a dados históricos é executado de maneira um pouco confusa, talvez porque o modelo visa primordialmente tempo de transação, que não é adequado a este tipo de operação.

No modelo de Su o tratamento de alterações retroativas não é detalhado, sendo provavelmente executado através da utilização de regras. Em Klopprogge, operações em dados válidos no passado são permitidas apenas a um tipo especial de usuário (*referee*) e causam perda de informações. No modelo de Käfer$_2$ menciona-se apenas atualizações ao estado corrente, não se discutindo alterações retroativas. Ariav afirma que em seu modelo informações passadas nunca são removidas. Entretanto, o tratamento de correções e atualizações não é especificado na descrição do modelo. Por fim, os modelos de Clifford, Tansel, Navathe, Wuu, Segev e Gabbay não tratam da questão de alterações.

2.2.9 Operadores Algébricos e Linguagens de Consulta

A maioria dos modelos apresentados introduz uma álgebra, uma linguagem de consulta ou ambos para a manipulação de suas construções temporais. Apenas os modelos de Lum, Klopprogge e Abbod não propõem nem uma álgebra nem uma linguagem de consulta. As álgebras apresentadas são extensões à álgebra relacional e divergem tanto na maneira como tratam os operadores relacionais tradicionais ($\cup$, $-$, $\times$, σ e π), quanto na introdução de novos operadores específicos para lidar com a dimensão temporal. Algumas das linguagens de consulta propostas são baseadas na própria álgebra estendida do modelo, enquanto outras são desenvolvidas de maneira independente. Nas subseções seguintes são analisadas as álgebras e linguagens de consulta dos diversos modelos aqui expostos.

Álgebras

Os operadores relacionais tradicionais são tratados de diversas maneiras nas álgebras dos vários modelos. Snodgrass, Jensen e Gabbay mantém os operadores sem alterações, sendo aplicáveis então apenas sobre instantâneos das relações. Da mesma forma, Sarda e Lorentzos mantém os operadores tradicionais sem alterações, porém os atributos de marcação de tempo são vistos como atributos normais, podendo ser acessados diretamente pelos operadores.

Gadia altera ligeiramente os operadores tradicionais para garantir a homogeneidade das tuplas e para considerar seus intervalos de validade. A aplicação dos operadores modificados sobre uma relação equivale à aplicação dos operadores originais em cada instante do *lifespan* da relação. Na

generalização de seu modelo os operadores são novamente modificados pois o requerimento de homogeneidade é retirado.

Clifford não altera os operadores $\cup$, $-$, $\cap$ e $\times$, mantendo-os aplicáveis diretamente sobre as relações temporais, todavia cria os operadores $\cup_o$, $\cap_o$ e $-_o$ que realizam a operação tradicional e em seguida juntam as tuplas com mesma chave que não se contradigam em algum instante de tempo. O operador σ é dividido em σ_{IF} e σ_{WHEN}. O operador σ_{IF} corresponde ao operador de seleção tradicional expandido com a especificação dos intervalos de tempo em que o predicado deve ser satisfeito e de um quantificador ($\exists$ ou $\forall$) para dizer se o predicado deve ser satisfeito em algum ou em todos os instantes de tempo dos intervalos. σ_{IF} recupera tuplas com o seu *lifespan* original. σ_{WHEN} tem como argumento uma condição sobre os dados e retorna as tuplas que satisfazem a condição, com seus *lifespans* restritos aos instantes em que a condição é satisfeita.

No modelo de Tansel, os operadores π, σ e $\times$ não são modificados; $\cup$ e $-$ sofrem pequenas alterações para lidar com tuplas iguais com intervalos que se intersectem. Ariav amplia os operadores σ e π gerando as operações de seleção de objetos e projeção, que equivalem a restrições nas dimensões de atributos e objetos em seus cubos de dados. Os operadores $\cup$, $-$ e $\times$ não são tratados no modelo.

As álgebras também divergem na especificação dos operadores construídos especificamente para manipular a nova dimensão temporal. Sarda define dois operadores principais para lidar com a semântica especial das relações históricas: COALESCE, que combina tuplas iguais com intervalos adjacentes ou com interseção, e EXPAND, que transforma cada tupla válida para um intervalo em um conjunto de tuplas, uma para cada instante do intervalo. É criado ainda o operador produto concorrente, que atua como um produto cartesiano, contudo emparelhando apenas as tuplas cujos intervalos de validade se intersectam.

Os dois operadores adicionais mais importantes de Lorentzos, FOLD e UNFOLD, transformam relações marcadas por pontos em relações marcadas por intervalos e vice-versa. FOLD transforma cada conjunto de tuplas iguais e válidas por pontos de tempo consecutivos em uma única tupla e a operação inversa é realizada pelo operador UNFOLD.

No modelo de Tansel são definidos os operadores TRIPLET-DECOMPOSITION e TRIPLET-FORMATION para transformar os intervalos de validade dos atributos em atributos normais e vice-versa. PACK e UNPACK são os principais operadores introduzidos por Tansel. O operador PACK aplicado sobre um atributo junta em uma tupla todas as tuplas cujas diferenças residam apenas nos valores deste atributo. O operador UNPACK aplicado sobre um atributo A cria uma tupla para cada valor assumido por A, repetindo os valores dos outros atributos quando for o caso. Na extensão do modelo para o relacional aninhado, em lugar dos operadores PACK e UNPACK são designados os operadores NEST e UNNEST.

É fácil observar a similaridade das abordagens de Sarda, Lorentzos e Tansel. De fato, a dimensão temporal é manipulada nestes modelos com o auxílio de operações do modelo relacional aninhado, implícita ou explicitamente. O tempo, na realidade, não chega a ser tratado como uma nova dimensão dos dados.

Jensen utiliza o operador TIME-SLICE, equivalente à seleção de BD instantâneo, para lidar com tempo de transação. Do mesmo modo, Snodgrass utiliza os operadores $\rho/\hat{\rho}$ para a seleção de um estado instantâneo/histórico de uma relação de tempo de transação/bitemporal. Sobre o estado histórico atuariam os operadores $\hat{\cup}$, $\hat{-}$, $\hat{\times}$, $\hat{\pi}$ e $\hat{\sigma}$, que são a contraparte histórica dos operadores tradicionais. O significado destes operadores não é formalizado nos artigos de descrição da álgebra acessíveis durante a execução desta revisão [MS87, MS90].

Os modelos de Jensen e Snodgrass efetuam a operação denominada *rollback*. Tal operação é aplicada sobre a dimensão do tempo de transação e serve para retornar o BD ou a relação ao seu estado conhecido em um momento qualquer do tempo passado.

No modelo de Gadia introduz-se um novo operador para retornar o *lifespan* de uma relação e outro para restringir uma relação aos dados relativos a um elemento temporal (seleção temporal). No modelo generalizado o operador de seleção temporal é substituído pelo operador WHEN (ω), que permite que os valores de um conjunto dos atributos de uma relação sejam restritos aos valores concernentes a um elemento temporal, calculado para cada tupla. São ainda possíveis restrições de elementos e atribuições temporais de n-dimensões para uma dimensão (veja o modelo de Gadia para os conceitos e atribuição e elemento temporal).

Clifford introduz dois novos operadores para dar suporte à dimensão temporal. O operador WHEN (Ω) retorna o conjunto de instantes de tempo para o qual uma tabela é definida. O operador TIME-SLICE estático ($\tau_{\Omega L}$) restringe uma tabela aos dados relativos a um intervalo de tempo, enquanto o operador TIME-SLICE dinâmico ($\tau_{\Omega A}$) restringe individualmente o *lifespan* de cada tupla para o valor de um atributo que expresse valores de tempo. É interessante observar que o operador σ_{WHEN} na realidade corresponde a um TIME-SLICE estático, que restringe o *lifespan* da tabela em questão ao conjunto de intervalos de tempo para os quais a condição é satisfeita.

Ariav define o operador de seleção temporal para manipular a dimensão de tempo. A seleção temporal efetua uma restrição nos dados de uma relação delimitando as fronteiras de tempo inferior e superior de acordo com uma dimensão temporal específica.

Pode-se observar que os operadores temporais de Gadia, Clifford e Ariav realizam o que poderíamos chamar de **fatiamento temporal**[2], isto é, restringem os dados a serem utlizados àqueles válidos em intervalo (Ariav e Clifford) ou conjunto de intervalos (Clifford e Gadia) de tempo, com relação a uma dimensão temporal específica. A operação *rollback* dos modelos de Snodgrass e Jensen também pode ser vista como um fatiamento temporal aplicado sobre o tempo de transação, restringindo os dados a um único instante. Após uma operação de *rollback* geralmente pode-se efetuar outro fatiamento temporal sobre os dados resultantes, levando-se em conta outra dimensão do tempo que não a de tempo de transação.

Visando completeza histórica, Gabbay define os operadores SINCE-PRODUCT e UNTIL-PRODUCT baseados na lógica US (*until-since*), que, segundo os autores, é totalmente expressiva para um modelo histórico, da mesma forma que a lógica de primeira ordem o é para um modelo de dados não temporal [GM91]. Com base nestes operadores, Gabbay define operadores mais amigáveis para selecionar dados que satisfaçam uma condição no BD anterior, em algum dos BD's anteriores ou em todos os BD's anteriores ao instante de referência da consulta e operadores análogos com relação a instantes futuros. Através da especificação do instante de referência da consulta e da utilização dos novos operadores, torna-se possível a realização da operação de fatiamento temporal.

Segev define alguns operadores de recuperação para suas TS's e cita a existência de alguns outros para definição e atualização de dados e operações sobre conjuntos. Os operadores de recuperação são compostos de três partes, a saber, **especificação de destino**, para indicar os pontos de dados da *time sequence collection* (TSC) de destino, **mapeamento**, para determinar os pontos das TSC's de origem a serem manipulados para gerar os pontos de destino, e a **função** a ser aplicada aos pontos de origem para gerar os pontos de destino. Os cinco operadores do modelo são obtidos por combinações de restrições nestas três partes; um operador genérico é criado para permitir ao

[2] Em inglês, *time slice*.

usuário a definição completa destas três partes. Embora o modelo seja mapeado para o relacional, não há a definição de uma álgebra ou linguagem de consulta.

Navathe define dois tipos de JOINS temporais e um operador para juntar tuplas iguais válidas em intervalos de tempo adjacentes. Entretanto, não apresenta uma álgebra temporal consistente para o modelo.

Lum, Klopprogge, Adiba, Abbod, Käfer, $Käfer_2$, Su e Wuu não propõem uma álgebra para seus modelos.

Linguagens de Consulta

Alguns autores introduzem a linguagem de consulta de seu modelo de acordo com a álgebra do próprio modelo. Assim procedem Gabbay, Sarda, Gadia e Ariav. Gabbay estende a linguagem SQL ampliando a cláusula FROM para possibilitar a utilização dos novos operadores temporais. Na extensão de Sarda do SQL, é introduzida a cláusula EXPAND BY e as cláusulas SELECT e FROM são aumentadas, respectivamente, com as opções COALESCED e CONCURRENT para suportar os novos operadores da álgebra do modelo. Uma nova cláusula (FROMTIME ... TOTIME ...) é acrescentada ao modelo para executar a operação de fatiamento temporal por intervalos, operação esta que não possui um operador correspondente na álgebra. A operação de *rollback* também é permitida no modelo, embora não esteja contida em sua álgebra histórica.

Gadia apresenta HTQuel, uma extensão à linguagem Quel, como linguagem de consulta de seu modelo inicial. São introduzidas as cláusulas TDOM e DURING, equivalentes respectivamente à recuperação dos períodos de validade de uma tabela e ao fatiamento temporal por conjuntos de intervalos de uma tabela. A cláusula DURING serve ainda para especificar alterações retroativas, sendo substituída pela cláusula EFFECTIVE em alterações pré-ativas. Gadia não apresenta uma linguagem de consulta para seu modelo estendido para o suporte de um número arbitrário de dimensões temporais.

A linguagem TOSQL, introduzida por Ariav, amplia o significado das cláusulas WHERE e SELECT para corresponderem aos operadores de seleção de objetos e projeção, respectivamente. A operação de seleção temporal (equivalente a um fatiamento temporal) é realizada usando-se uma das novas cláusulas criadas com este propósito (AT, WHILE, DURING, BEFORE, AFTER). Uma nova cláusula (AS-OF) é especificada para efetuar a operação de *rollback*, embora esta operação não seja mencionada na descrição dos operadores do modelo. As cláusulas de seleção temporal e AS-OF possuem a opção ALONG para designar a dimensão temporal utilizada para realizar suas operações.

Para o modelo de Navathe é definida a linguagem TSQL — uma extensão à linguagem SQL. Os principais incrementos à linguagem SQL são as inclusões das cláusulas TIME-SLICE, que realiza um fatiamento temporal por intervalos, e WHEN, que é usada para especificar predicados temporais para seleção de tuplas de acordo com o período de validade destas.

As principais extensões à linguagem Quel incluídas em TQuel — a linguagem de consulta do modelo de Snodgrass — são as cláusulas WHEN, AS-OF e VALID. A cláusula WHEN é usada para especificar condições para seleção de tuplas de acordo com os seus períodos de validade; AS-OF realiza a operação de *rollback* e VALID especifica os valores de tempo em relações resultantes de consultas ou para alterações no BD. A cláusula WHEN pode ser utilizada ainda em alterações retroativas e pré-ativas.

O modelo de Adiba possui uma linguagem de consulta estendida a partir da linguagem

LAMBDA (por sua vez uma extensão do SQL). A maior alteração à linguagem é a possibilidade de indicação das versões de um histórico a serem recuperadas ou corrigidas através das posições ordinais destas versões ou da data de sua criação.

Käfer introduz uma linguagem baseada no SQL para manipulação de suas TS's. São introduzidas cláusulas para a criação, atualização correção, remoção lógica ou física e recuperação de dados em TS's. Nas quatro primeiras operações o valor do tempo válido deve ser fornecido pelo usuário. Em todas as operações pode ser especificada a cláusula T_WHERE, que realiza uma seleção verificando-se a validade de um predicado em algum ou em todos os pontos de tempo de um conjunto de instantes de tempo. Este conjunto é determinado pela indicação explícita de um instante ou intervalo ou por uma condição sobre os dados. Dadas as TS's que satisfaçam à cláusula T_WHERE, a operação de recuperação permite ainda especificar se serão recuperadas todas as suas tuplas, apenas as tuplas que satisfaçam à cláusula ou as tuplas válidas em um instante ou intervalo de tempo específico.

Su estende uma linguagem de consulta denominada OQL para o suporte a dados temporais. A alteração mais importante efetuada sobre a linguagem é a inclusão da cláusula WHEN que realiza o fatiamento temporal para um intervalo de tempo designado diretamente ou para um conjunto de intervalos de tempo determinado por uma condição sobre os dados. Tansel introduz uma linguagem gráfica (TBE) para seu modelo, linguagem esta que, segundo os autores, possui a mesma expressividade da álgebra do modelo.

Wuu utiliza a linguagem original do modelo OODAPLEX para seu modelo temporal. As consultas temporais são possibilitadas pelo acesso direto aos *timestamps*. Clifford, Lum, Klopprogge, Segev, Abbod, Lorentzos e Jensen não mencionam o desenvolvimento de linguagens de consulta para seus modelos.

2.2.10 Quadros Sinópticos

A seguir apresentamos dois quadros sinópticos que sumarizam as características analisadas nesta seção. O tópico **Evolução de Esquema** indica apenas se esta questão foi ou não considerada nos diversos modelos, o que não significa que os modelos tratem esta questão de maneira satisfatória. **Preservação de Informações** exprime se os modelos permitem a manutenção de todas as informações já introduzidas no BD, mesmo após alterações retroativas. O item **Alterações Retroativas** indica os modelos capazes de realizar modificações sobre dados válidos no passado. Estas duas propriedades só serão verificadas nos modelos que lidam com modificações.

2.3 Considerações Finais

O número de trabalhos publicados sobre o tema BDT's denota a importância com que esta área está sendo considerada pela comunidade científica. No entanto, ainda existem várias lacunas a serem preenchidas. As diferenças existentes entre as diversas propostas demonstram que não existe consenso sobre várias questões concernentes a BDT's. Além disto, algumas questões, tais como evolução de esquema e convivência de diversos tipos de dados temporais, foram pouco estudadas. Por fim, trabalhos que visem modelos de dados que não o relacional são ainda em pequeno número. Os modelo e linguagem apresentados nos capítulos a seguir constituem uma tentativa de cooperar na evolução do entendimento do problema temporal.

Tabela 2.1:

	Modelo	Dimensões Temporais	Variação de Valores	Álgebra ou Linguagem	Evolução de Esquema
Clifford	Relacional	Válido	Escada	Álgebra	√
Klopprogge	ER	Transação	Definida pelo Usuário	Nenhuma	○
Lum	Relacional	Transação[1]	Escada	Nenhuma	√
Snodgrass	Relacional	Válido/Transação	Escada/Discreta	Ambas	√
Tansel	Relacional	Válido	Escada	Álgebra	○
Gadia	Relacional	Arbitrário[1]	Escada	Ambas	○
Adiba	ER	Transação	Escada	Linguagem	√
Ariav	Relacional	Arbitrário	Escada	Ambas	○
Segev	Independente[1]	Válido	4 tipos	Álgebra[1]	○
Navathe	Relacional	Válido[1]	Escada	Linguagem[1]	√
Abbod	Relacional	Válido/Transação	Escada	Nenhuma	○
Lorentzos	Relacional	Válido	Escada/Discreta	Álgebra	○
Sarda	Relacional	Válido/Transação	Escada/Discreta	Ambas	○
Käfer	Relacional	Válido/Transação[1]	4 tipos	Linguagem	○
Käfer_2	Objetos Complexos	Válido[1]	Escada	Linguagem	○
Gabbay	Relacional	Válido	Escada	Ambas	√
Jensen	Relacional	Transação	Escada	Álgebra	○
Su	OO	Válido[1]	Escada	Linguagem	○
Wuu	OO	Arbitrário	Escada	Linguagem	○

√ Satisfaz critério
○ Não satisfaz critério
[1] Veja texto

Tabela 2.2:

	Nível de Registro de Tempo	Representação do Tempo	Implementação	Preservação de Informações	Alterações Retroativas
Clifford	Atributo[1]	Pontos[1]	○	NA	NA
Klopprogge	Atributo	Pontos	√	○	√
Lum	Tupla	Intervalos[1]	∂	√	√
Snodgrass	Tupla	Pontos/Intervalos	√	√	√
Tansel	Atributo	Intervalos[1]	√	NA	NA
Gadia	Atributo	Conjunto de Interv.	○	○	√
Adiba	Variável	Pontos	∂	○	√
Ariav	Tupla	Pontos	∂	NA	NA
Segev	Tupla	Pontos	○[1]	NA	NA
Navathe	Tupla	Intervalos	○	NA	NA
Abbod	Tupla	Pontos	√	√	√
Lorentzos	Tupla	Pontos/Intervalos	√	○	√
Sarda	Tupla	Intervalos	√	√	√
Käfer	Tupla	Pontos	∂	√	√
Käfer_2	Objeto	Intervalos	∂	√	○
Gabbay	Tupla	Intervalos	○	NA	NA
Jensen	Tupla	Pontos	∂	√	○
Su	Objeto	Intervalos	∂	○	√
Wuu	Variável	Intervalos	∂	NA	NA

√ Satisfaz critério
○ Não satisfaz critério
∂ Implementação em progresso ou mapeamento descrito
NA Não analisado pelos autores
[1] Veja texto

Capítulo 3

Modelo de Dados Temporal TOODM

Este capítulo descreve o modelo temporal TOODM (*Temporal Object Oriented Data Model*). No modelo são consideradas as questões de tipos de tempo, tipos de variações de valores, granularidade dos valores de tempo, identidade de objetos e manutenção de diversos esquemas.

O modelo oferece suporte a tempo válido e tempo de transação, como descritos na seção 1.3.1. Como resultado disto, existirão quatro categorias temporais de classes — instantânea, de tempo válido, de tempo de transação e bitemporal. Além disto, o modelo suporta classes com variações de valores de quatro tipos distintos — escada, discreta, contínua e definida pelo usuário — e permite a utilização de diversas granularidades para marcação do tempo. O modelo apresenta um processo para compatibilizar as características temporais das classes que se relacionem por herança e/ou composição. O modelo trata ainda das questões de migração de objetos e evolução de esquema considerando todas as características temporais das classes e objetos envolvidos.

O capítulo divide-se da seguinte forma. A seção seguinte descreve as categorias temporais das classes e o relacionamento entre elas. A seção 3.2 introduz o processo de temporalização, considerando apenas as diferenças nas categorias temporais das classes. A seção 3.3 introduz a questão da variação de valores no processo de temporalização e a seção 3.4 considera a questão da granularidade. A seção 3.5 sumariza o processo de temporalização, enquanto na seção 3.6 descrevemos o tratamento das classes surgidas com a temporalização. A seção 3.7 trata da identidade de objetos. A manutenção de diversos esquemas é tratada na seção 3.8.

3.1 Categorias Temporais

Nosso modelo lida com as dimensões de tempo válido e tempo de transação. Por este motivo tivemos que optar entre duas estratégias em relação aos tipos de classes possíveis. A primeira opção, mais restritiva, seria permitir somente a existência de classes com suporte simultâneo a tempo válido e de transação, ou seja, com características semelhantes a um BDBT. A segunda opção, mais flexível, é a adotada na dissertação: são permitidas classes com suporte apenas a tempo válido, apenas a tempo de transação, a ambos os tipos de tempo ou sem suporte a tempo. Assim possibilitamos ao usuário a escolha dos tipos de classes de forma adequada a suas necessidades. Assim poderá existir simultaneamente no BD, classes com características dos quatro tipos de BD's descritos na seção 1.3:

Classes instantâneas: São classes tradicionais, sem suporte a tempo.

Classes de tempo válido: Seus objetos são capazes de expressar quando os dados são válidos no mundo real, porém não guardam os estados do BD como conhecidos no passado. Podem armazenar e alterar dados relativos ao passado e ao futuro, mantendo-os sempre como conhecidos no presente.

Classes de tempo de transação: Seus objetos mantêm estados passados do BD, contudo não são capazes de indicar quando os dados são válidos no mundo real. É possível recuperar estados de objetos como conhecidos no passado.

Classes bitemporais: Seus objetos mantêm estados passados do BD ao mesmo tempo em que guardam os períodos de tempo de validade de cada dado. Permitem armazenar, alterar e recuperar dados relativos ao passado e ao futuro como conhecidos no presente bem como de recuperar dados como conhecidos no passado.

O modelo dispõe portanto desde classes sem suporte a tempo até classes com suporte aos dois tipos de tempo. Denominamos **categoria temporal** de uma classe sua classificação como instantânea, de tempo válido, de tempo de transação ou bitemporal.

Com base nestes conceitos chegamos a uma definição para BDT: *Um BDT é definido como a seqüência de todos os estados históricos do BD, do momento de sua criação até o presente*. Cada estado histórico E_j do BD consiste no esquema conservado do BD no tempo de transação t_j e nos dados mantidos do BD como conhecidos neste t_j.

Cada um dos estados históricos passados contém apenas dados e esquema relativos a classes de tempo de transação e bitemporais, enquanto o estado correspondente ao tempo presente contém também dados e esquema de classes instantâneas e de tempo válido. Isto acontece porque, da mesma forma que os BD's vistos na seção 1.3, as classes de tempo válido e instantâneas correspondem a conhecimento atual, enquanto as classes de tempo de transação e bitemporais também expressam conhecimento passado. Portanto, os esquemas de estados passados só podem conter dados de classes de tempo de transação e bitemporais (as únicas com estados anteriores armazenados). Para o estado do BD no presente, é mantido o esquema com os quatro tipos de classes.

Esta definição desobriga o BDT de possuir um esquema imutável. O esquema pode evoluir com o tempo, contanto que, em cada instante referente ao tempo de transação, um esquema esteja associado ao BD e os dados estejam coerentes com este esquema. Em outras palavras, podem ser mantidos vários esquemas para o BD, cada qual associado ao período de tempo de transação em que foi ativo. A questão de evolução de esquema é tratada na seção 3.8.3.

A evolução temporal de um BDT pode ser vista da seguinte forma. Suponha que em um BDT no estado presente existam quatro classes: C_I (instantânea), C_{TV} (de tempo válido), C_{TT} (de tempo de transação) e C_{BT} (bitemporal). Cada uma destas classes tem uma extensão (ou seja, um conjunto de objetos). Suponha que certas alterações sejam feitas em algum objeto de alguma classe, o que significa a criação de um novo estado com marca de tempo de transação. O BD terá agora um novo estado, com as classes C_I, C_{TV}, C_{TT} e C_{BT} que são as cópias do estado anterior destas classes com as modificações realizadas na alteração. Do estado anterior, no entanto, só permanecerão armazenados os estados de C_{TT} e C_{BT}. As duas outras classes só existem no presente.

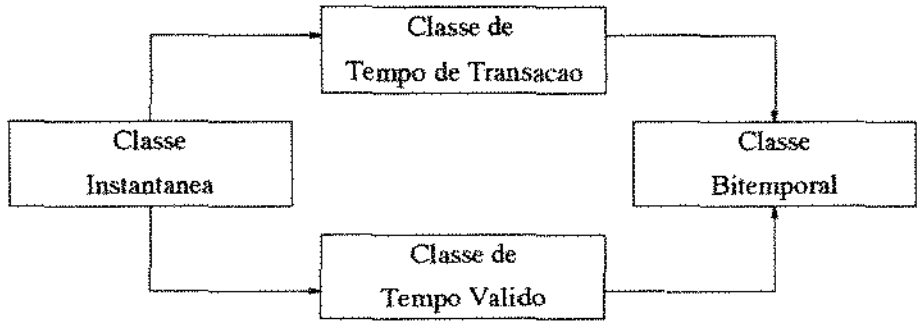

Figura 3.1: Ordem parcial das classes conforme seu tipo temporal.

3.2 Temporalização e Categorias Temporais

Estabelecemos na seção anterior que existirão classes de quatro categorias temporais distintas. Todavia, as classes pertencentes a um BD possuem ligações de herança e composição entre si. Logo, deve haver uma compatibilização entre as características temporais das classes. Denominaremos este processo de compatibilização de **temporalização**. O processo de temporalização deve ser executado automaticamente no momento da criação do esquema do BD e sempre que ocorrerem alterações nele.

As classes de cada uma das quatro categorias temporais aceitam operações temporais diferentes, conforme o tipo de tempo que suportam. Pode-se, então, estabelecer uma ordem parcial entre as categorias temporais, onde uma classe de uma categoria temporal é capaz de aceitar ao menos as operações temporais possíveis em classes das categorias temporais antecessoras. Esta ordem parcial pode ser vista na figura 3.1.

Esta ordem indica, por exemplo, que uma classe bitemporal permite qualquer operação temporal possível nas outras classes e que uma classe de tempo válido aceita as operações temporais possíveis em uma classe instantânea, porém nem todas as operações de classes de tempo de transação e bitemporais. O mecanismo de temporalização será baseado nesta ordem parcial entre as classes.

A temporalização de classes que já contenham objetos constitui um problema de evolução de esquema, agravado pela manutenção de estados passados. Trataremos deste caso na seção sobre a evolução de esquema (seção 3.8.3); nesta seção e nas seguintes tratamos apenas da temporalização de classes vazias.

A temporalização, embora seja um processo atômico, é realizada em duas fases: primeiramente compatibilizando as classes pelo grafo de herança e a seguir pelo grafo de composição.

3.2.1 Compatibilização Temporal — Herança

Na temporalização por herança as subclasses são compatibilizadas com as superclasses. Pelo paradigma da OO, um objeto pertencente a uma classe C pode ser substituído em qualquer circunstância por um objeto da classe C_1, sendo C uma superclasse de C_1. Isto traz como conseqüência a necessidade de que a classe C_1 tenha, ao menos, as mesmas propriedades temporais da classe C. Assim, ao se criar uma subclasse, ela deve ser temporalizada para compatibilizar suas propriedades temporais com as de suas superclasses.

As subclasses são alteradas segundo a ordem parcial definida acima, para incorporarem as características da superclasse às suas características originais.

C	Instantânea	de Tempo Válido	de Tempo de Transação	Bitemporal
C_1	Instantânea	de Tempo Válido	de Tempo de Transação	Bitemporal
C_2	de Tempo Válido	de Tempo Válido	Bitemporal	Bitemporal
C_3	de Tempo de Transação	Bitemporal	de Tempo de Transação	Bitemporal
C_4	Bitemporal	Bitemporal	Bitemporal	Bitemporal

Tabela 3.1: Esquema de Temporalização

Considerando uma classe C e suas subclasses C_1, C_2, C_3 e C_4, declaradas originalmente instantânea, de tempo válido, de tempo de transação e bitemporal, respectivamente, a tabela 3.1 resume quais seriam os novos tipos das subclasses após a temporalização, dependendo do tipo declarado da classe C. A segunda coluna, por exemplo, mostra que se C for de tempo válido, então a temporalização transformará C_1 em classe de tempo válido e C_3 em bitemporal.

Exemplo:

Suponha o esquema de um BD com uma classe *PESSOA* definida como de tempo válido:

Classe *PESSOA* de tempo válido.

Se agora definirmos a classe *EMPREGADO*:

Classe *EMPREGADO* instantânea herda de *PESSOA*.

A temporalização destas classes criaria uma nova classe, de tempo válido, *EMPREGADO* que substituiria a classe instantânea *EMPREGADO*. Note-se que, segundo o paradigma da OO todo empregado também é uma pessoa e, portanto, todo objeto da classe *EMPREGADO* deve ter as características da classe *PESSOA*. Desta forma, não faz sentido que a classe instantânea *EMPREGADO* continue a existir, visto que todas as instâncias de empregado pertencerão obrigatoriamente a *EMPREGADO* de tempo válido. Como uma classe de tempo válido é capaz de suportar todas as operações de uma classe instantânea, a classe *EMPREGADO* de tempo válido pode substituir a classe instantânea sem prejuízos. O sistema de *defaults* da linguagem de consulta do modelo permite que o usuário trate a classe *EMPREGADO* de tempo válido como uma classe instantânea sem que haja problemas (vide seção 4.7). O sistema de *defaults* age da mesma forma nos outros casos em que há conversões de classes com a temporalização.

□

Desta forma, na fase de temporalização por herança, quando houver a necessidade de alteração nas características temporais de uma classe, é criada uma nova classe com as capacidades temporais requeridas e a classe antiga é eliminada.

3.2.2 Compatibilização Temporal — Composição

A temporalização por composição é necessária porque, ao mantermos um objeto composto em um certo tempo, precisaremos manter informação temporal sobre os valores de todos os seus componentes neste tempo. Portanto, se uma classe C_1 é componente de uma classe C, a classe C_1 deve suportar ao menos os mesmos tipos de tempo da classe C — o que é garantido com a temporalização.

A temporalização por composição segue em linhas gerais o esquema de temporalização por herança. Na temporalização por composição as classes componentes são alteradas para se compatibilizarem com as classes compostas da mesma forma que as subclasses eram modificadas para ficarem de acordo com as superclasses. Note-se que as classes componentes herdadas devem ser temporalizadas da mesma forma que as demais classes componentes.

A tabela 3.1 pode ser utilizada para indicar as modificações após a temporalização por composição, considerando agora C uma classe composta pelas classes C_1, C_2, C_3 e C_4, definidas inicialmente como instantânea, de tempo válido, de tempo de transação e bitemporal, respectivamente.

Exemplo:

Considere as seguintes classes, definidas originalmente como:

Classe *PESSOA* **de tempo válido**
NOME : STRING;
PESO : INTEIRO.

Classe *EMPREGADO* **instantânea herda de** *PESSOA*
FUNÇÃO : STRING;
RESIDÊNCIA : ENDEREÇO.

Classe *ENDEREÇO* **instantânea**
CIDADE : STRING;
RUA : STRING;
NÚMERO : INTEIRO.

Como já vimos, a primeira fase de temporalização é efetuada por herança e substitui a classe *EMPREGADO* instantânea (que deixa de existir) pela classe *EMPREGADO* de tempo válido. Na segunda fase, executa-se a temporalização por composição, resultando deste processo o seguinte esquema:

Classe *PESSOA* **de tempo válido**
NOME : STRING **de tempo válido;**
PESO : INTEIRO **de tempo válido.**

Classe *EMPREGADO* **de tempo válido herda de** *PESSOA* **de tempo válido**
FUNÇÃO : STRING **de tempo válido;**
RESIDÊNCIA : ENDEREÇO **de tempo válido.**

Classe *ENDEREÇO* **de tempo válido**

CIDADE : STRING **de tempo válido;**
RUA : STRING **de tempo válido;**
NÚMERO : INTEIRO **de tempo válido.**

Classe *ENDEREÇO* **instantânea**
CIDADE : STRING **instantânea;**
RUA : STRING **instantânea;**
NÚMERO : INTEIRO **instantânea.**

Na temporalização por composição surge uma nova classe *ENDEREÇO* de tempo válido, mas a classe *ENDEREÇO* instantânea é mantida.

□

Aqui há dois fatos a serem observados. As classes básicas providas pelo sistema (como *STRING* e *INTEIRO*) são originalmente do tipo instantâneo. Isto permite que estas classes possam assumir qualquer comportamento temporal como componentes de outras classes. Se, por exemplo, as classes originais do sistema fossem de tempo de transação, não seria permitido que um objeto de uma destas classes tivesse características estritamente instantâneas. Obviamente, uma classe de tempo de transação pode executar todas as operações possíveis em uma classe instantânea, contudo obriga a manutenção de estados passados, o que pode não ser desejado pelo usuário.

Na temporalização por composição não se eliminam as classes cujo comportamento temporal foi modificado — no exemplo acima podemos notar a permanência da classe *ENDEREÇO* instantânea no esquema final.

A razão disto é que nem todos objetos que se queira definir como *ENDEREÇO* serão constituintes da classe *EMPREGADO* — *ENDEREÇO* pode ser componente de outras classes. Isto fica mais claro ao considerarmos a classe *STRING*. A criação de uma classe *STRING* de tempo válido não inviabiliza a existência da classe *STRING* instantânea pois, se isto acontecesse, ao temporalizarmos a classe *ENDEREÇO* de tempo válido, alterando o tipo de *CIDADE* de *STRING* instantânea para *STRING* de tempo válido, obrigaríamos todos os objetos do tipo *STRING* a ter características de uma classe de tempo válido, o que evidentemente não era o efeito desejado.

3.2.3 Compatibilização "Recursiva"

Embora possa parecer à primeira vista que as classes do BD devam estar totalmente compatíveis após a execução da temporalização como a definimos acima, ainda podem existir alguns problemas. Ilustramos isto com o exemplo abaixo.

Exemplo:

Consideremos que no esquema do exemplo anterior tenhamos também as seguintes classes:

Classe *REVISTA* **bitemporal**
TÍTULO : STRING;
EDITOR : PESSOA;

NÚMERO : INTEIRO;
PREÇO : DINHEIRO.

Classe *DINHEIRO* de tempo de transação
MOEDA : STRING;
VALOR : REAL.

Como vimos na seção anterior, as classes componentes de *REVISTA* serão compatibilizadas com as características temporais desta, durante a temporalização por composição. Em particular, a compatibilização da classe *PESSOA* (definida como de tempo válido) provocará a criação de uma classe *PESSOA* bitemporal. Portanto, instâncias de *PESSOA* que fossem componentes de *REVISTA* pertencerão à classe *PESSOA* bitemporal e serão compatíveis com a classe *REVISTA* bitemporal. Todavia, seguindo o paradigma OO, objetos de *EMPREGADO* poderiam substituir objetos de *PESSOA* como editores de revista. Mas não existe a classe *EMPREGADO* bitemporal, e objetos de *EMPREGADO* de tempo válido não podem substituir instâncias de *PESSOA* bitemporal!

□

Concluímos do exemplo acima que as classes devem ser compatibilizadas novamente por suas ligações de herança após a compatibilização por composição. Esta segunda compatibilização por herança tem como objetivo compatibilizar as subclasses com as superclasses originadas na temporalização por composição (no exemplo acima, compatibilizar *EMPREGADO* com *PESSOA* bitemporal, surgida na temporalização de *REVISTA*).

Note que existe uma diferença entre a primeira e a segunda temporalização por herança. Na primeira temporalização, a classe *EMPREGADO* de tempo válido **substitui** uma classe já existente (*EMPREGADO* instantâneo) como subclasse de *PESSOA* de tempo válido. Na segunda temporalização, ocorre apenas a **criação** de uma nova classe (*EMPREGADO* bitemporal) que será subclasse de *PESSOA* bitemporal, surgida na temporalização por composição. A classe *EMPREGADO* bitemporal surgiu para reparar um problema de consistência ocasionado pela temporalização: a classe *PESSOA* bitemporal não possuía uma subclasse correspondente à relação de herança *PESSOA* *EMPREGADO*.

Na segunda temporalização por herança então, não são eliminadas classes. Ocorre apenas a criação das subclasses correspondentes às classes originadas na temporalização por composição.

Com o surgimento de novas classes na segunda compatibilização por herança, o grafo de composição deverá ser novamente temporalizado. É fácil notar a necessidade da nova compatibilização observando o exemplo corrente. A criação de *EMPREGADO* bitemporal implicará na temporalização de suas componentes (*STRING* e *ENDEREÇO*), e das componentes destas sucessivamente.

Novamente, seriam criadas novas classes na temporalização por composição, que poderiam exigir outra compatibilização por herança, e assim sucessivamente. Portanto, o processo de compatibilização (por herança e composição) deve ser executado repetidas vezes, até que o esquema não mais seja alterado. Podemos assegurar que o processo não se repetirá indefinidamente já que o grafo de herança é finito e não contém ciclos, e a cada passo de compatibilização por herança/composição garantimos que um nível do grafo de herança não será mais afetado.

Exemplo:

Mostramos abaixo o esquema final de nosso exemplo corrente, resultante do processo de temporalização.

Classe *REVISTA* **bitemporal**
TÍTULO : STRING **bitemporal;**
EDITOR : PESSOA **bitemporal;**
NÚMERO : INTEIRO **bitemporal;**
PREÇO : DINHEIRO **bitemporal.**

Classe *DINHEIRO* **bitemporal**
MOEDA : STRING **bitemporal ;**
VALOR : REAL **bitemporal.**

Classe *DINHEIRO* **de tempo de transação**
MOEDA : STRING **de tempo de transação ;**
VALOR : REAL **de tempo de transação.**

Classe *PESSOA* **bitemporal**
NOME : STRING **bitemporal;**
PESO : INTEIRO **bitemporal.**

Classe *PESSOA* **de tempo válido**
NOME : STRING **de tempo válido;**
PESO : INTEIRO **de tempo válido.**

Classe *EMPREGADO* **bitemporal herda de** *PESSOA* **bitemporal**
FUNÇÃO : STRING **bitemporal;**
RESIDÊNCIA : ENDEREÇO **bitemporal.**

Classe *EMPREGADO* **de tempo válido herda de** *PESSOA* **de tempo válido**
FUNÇÃO : STRING **de tempo válido;**
RESIDÊNCIA : ENDEREÇO **de tempo válido.**

Classe *ENDEREÇO* **bitemporal**
CIDADE : STRING **bitemporal;**
RUA : STRING **bitemporal;**
NÚMERO : INTEIRO **bitemporal.**

Classe *ENDEREÇO* **de tempo válido**
CIDADE : STRING **de tempo válido;**
RUA : STRING **de tempo válido;**
NÚMERO : INTEIRO **de tempo válido.**

Classe *ENDEREÇO* **instantânea**
CIDADE : STRING **instantânea**;
RUA : STRING **instantânea**;
NÚMERO : INTEIRO **instantânea**.

□

A temporalização é um processo atômico, sendo dividida em duas fases de compatibilização, que poderão ser repetidas diversas vezes. O tratamento dispensado às diversas classes originadas na temporalização é explicado na seção 3.6.

3.3 Variação de Valores

Por simplicidade, não consideramos a questão de variação de valores no tempo nas seções anteriores. Todavia, como vimos na seção 1.3.2, a semântica do tempo válido requer que se defina como é feita a variação dos valores dos objetos no decorrer do tempo (variação escada, discreta, etc). Portanto, as classes com suporte a tempo válido (bitemporais e de tempo válido) devem especificar como é a variação dos valores de seus objetos em relação ao tempo.

O exemplo a seguir descreve como o SGBD determina os valores de um objeto de acordo com o tipo de variação da classe à qual ele pertence.

Exemplo:

Considere a criação de um objeto O em uma classe de tempo válido C em um instante t_1, atribuindo a este objeto o valor v_1. Se o usuário não especificar o tempo de validade de v_1, o SGBD lhe atribuirá um tempo de validade de acordo com o tipo de variação de valores de C. Caso a variação seja discreta, contínua ou definida pelo usuário, o SGBD determina, por *default*, que v_1 é válido apenas em t_1. Em caso de variação em escada, v_1 seria válido do instante t_1 até *now* — *now* representa o tempo corrente e, portanto, varia constantemente.

Considere agora que alteremos o valor de O para v_2, especificando a validade deste valor para o tempo t_2 (t_2 posterior a t_1). Se quisermos recuperar o valor de O em um tempo entre t_1 e t_2, a resposta obtida dependerá do tipo de variação de valores de C. No caso de variação em escada será retornado v_1. No caso da variação discreta, será retornado um valor nulo, pois não existiriam valores entre t_1 e t_2. Se a variação de C for contínua o SGBD utilizará uma função predefinida do próprio sistema para interpolar o valor no tempo procurado. Caso C apresente uma variação de valores definida pelo usuário, a interpolação será feita por uma função indicada pelo usuário.

□

Com a temporalização, novas classes de tempo válido e bitemporais devem surgir, e o tipo de variação de valores destas classes deve ser determinado durante o processo. Além disto, podem existir classes com tipo de variação de valores diferentes ligadas por laços de herança e/ou composição.

Utilizaremos o seguinte exemplo para ilustrar o tratamento destas questões.

Exemplo:

Considere as classes definidas abaixo:

Classe *REVISTA* **bitemporal discreta**
TÍTULO : STRING;
EDITOR : PESSOA;
NÚMERO : INTEIRO;
PREÇO : DINHEIRO **discreto.**

Classe *PESSOA* **de tempo válido escada**
NOME : STRING;
PESO : INTEIRO **contínuo.**

Classe *DINHEIRO* **de tempo de transação**
MOEDA : STRING;
VALOR : REAL.

Ao efetuarmos a temporalização, novas classes seriam criadas, resultando no seguinte esquema:

Classe *REVISTA* **bitemporal discreta**
TÍTULO : STRING **bitemporal discreta;**
EDITOR : PESSOA **bitemporal escada;**
NÚMERO : INTEIRO **bitemporal discreta;**
PREÇO : DINHEIRO **bitemporal discreto.**

Classe *PESSOA* **bitemporal escada**
NOME : STRING **bitemporal escada;**
PESO : INTEIRO **bitemporal contínuo.**

Classe *PESSOA* **de tempo válido escada**
NOME : STRING **de tempo válido escada;**
PESO : INTEIRO **de tempo válido contínuo.**

Classe *DINHEIRO* **de tempo de transação**
MOEDA : STRING **de tempo de transação;**
VALOR : REAL **de tempo de transação.**

Classe *DINHEIRO* **bitemporal discreta**
MOEDA : STRING **bitemporal discreta;**
VALOR : REAL **bitemporal discreto.**

□

É importante notar que as classes inicialmente declaradas com tipo de variação de valores diferentes (e.g., *REVISTA* e *PESSOA*) não precisam ser compatibilizadas, com respeito à variação, nem pelo grafo de herança nem pelo de composição.

No caso de **herança** não haveria a necessidade da temporalização porque, ao se declarar duas classes C e C_1 com tipos de variação distintos, sendo C superclasse de C_1, não se criaria nenhuma incompatibilidade entre objetos das duas classes. Todas as operações aplicáveis a objetos da classe C ainda seriam aplicáveis a objetos da classe C_1.

Com respeito à **composição** também não há necessidade de compatibilizarmos as classes com diferentes variações dos valores de seus objetos. Considere agora C_1 uma classe componente de C. A variação de valores da classe C_1 determinará como os valores dos objetos desta classe evoluirão com o decorrer do tempo, enquanto o tipo de variação da classe C determinará quando os objetos da classe C_1 estarão ligados à classe C.

Note que, como não há necessidade de que as classes composta e componentes apresentem o mesmo tipo de variação, a variação das classes componentes seguirá um dos seguintes casos:

1. O suporte a tempo válido na classe componente surgiu em decorrência da temporalização. Neste caso, no próprio processo de temporalização a classe componente será definida, por *default*, com o mesmo tipo de variação de valores da classe composta que a originou. Esta é a situação, por exemplo, das classes *STRING* bitemporal discreta (*TÍTULO*) e *REAL* bitemporal discreto (*VALOR*).

2. Quando a classe componente já tiver sido definida com suporte a tempo válido e, conseqüentemente, estiver com o tipo de variação de valores já definido, o processo de temporalização, por *default*, não causará nenhuma alteração, mesmo quando a classe composta tiver um tipo de variação diverso da classe componente. No exemplo acima a classe *PESSOA* com variação de valores em escada é definida como componente da classe *REVISTA*, com variação discreta. A classe *PESSOA* não precisará ser redefinida como discreta para se compatibilizar com a classe *REVISTA*. Como dissemos anteriormente, não há problemas neste caso — objetos da classe *PESSOA* estarão associados a objetos da classe *REVISTA* em pontos discretos de tempo, contudo seus valores variarão segundo uma função escada.

3. Esta situação ocorre quando não desejamos que o SGBD assuma o *default* em um dos casos anteriores. Neste caso, é preciso especificar o tipo de variação de valores desejado da classe componente na definição da variável de instância correspondente. Este é o caso da classe *INTEIRO* contínuo (*PESO*). Novamente, não há inconvenientes em uma classe com variação em escada (*PESSOA*) ter como componente uma classe com variação contínua (*INTEIRO* da componente *PESO*) — um objeto da classe *INTEIRO* pode estar associado a uma pessoa em um intervalo de tempo, contudo o valor deste objeto variará de forma contínua neste intervalo.

3.4 Granularidade

A granularidade dos valores de tempo utilizados para marcar os dados do BD é o terceiro aspecto em que as classes com suporte a tempo podem apresentar variações. Contudo, antes de discorrermos

a respeito da granularidade dos valores de tempo, introduziremos uma classificação dos tipos de valores de tempo existentes em nosso modelo.

Classificamos os valores de tempo em quatro grupos, a saber, pontos de tempo, *spans*, intervalos e elementos temporais. **Pontos de tempo** representam um instante único de tempo, de uma certa granularidade (e.g., uma data representa um dia específico na linha do tempo). Um **span**, embora seja representado como um ponto de tempo, indica uma quantidade de tempo (e.g. uma data passaria a representar uma quantidade de dias, meses e anos, ao invés de um dia específico). Um **intervalo** é o conjunto de todos os instantes de tempo entre dois pontos de tempo (inclusive). **Elementos temporais** são conjuntos de intervalos disjuntos. Elementos temporais apresentam vantagens sobre intervalos por serem mais adequados aos operadores de união, interseção e diferença de conjuntos [Gad88].

Estes valores de tempo são freqüentemente utilizados em um BDT. Além de serem usados para marcar os dados do BD, podem estar presentes em respostas a consultas, bem como na formulação destas. Podem ainda estar presentes em classes que representem tempo de usuário. A granularidade destes valores de tempo pode variar de anos a milissegundos e o SGBD deve lidar com estes valores de forma uniforme. A regra utilizada pelo SGBD para lidar com valores de diferentes precisões é trabalhar sempre com a maior precisão envolvida, i.e., tratar todos os valores de tempo envolvidos em cada operação convertidos para a granularidade mais fina existente entre os valores considerados. Esta conversão considera um valor de menor precisão como representando todos os instantes expressáveis numa granularidade mais fina. Por exemplo, um valor com precisão a nível de anos, se convertido a uma granularidade mensal englobaria todos os meses do ano; um ponto de tempo de granularidade anual tornar-se-ia um intervalo do primeiro ao último mês do ano quando convertido para a granularidade mensal.

Os operadores definidos para lidar com os quatro tipos de valores de tempo são expostos no capítulo 4. Neste capítulo trataremos apenas da definição da granularidade de marcação de tempo de classes temporais, e do relacionamento entre classes com granularidades distintas.

Como a determinação do tempo de transação é de responsabilidade exclusiva do SGBD, a granularidade dos valores relativos a tempo de transação independe do usuário e da aplicação. Desta forma, a granularidade para a marcação de tempo de transação será sempre a mesma em todas as classes com suporte a este tipo de tempo, para um dado SGBD. Contudo, a granularidade de tempo válido pode variar de classe para classe. Portanto, é necessário estabelecer quais os ajustes necessários para compatibilizar as classes com suporte a tempo válido, no aspecto de granularidade, durante a temporalização.

3.4.1 Herança

Sejam duas classes com suporte a tempo válido C e C_1 ligadas por herança, C superclasse de C_1, e denominemos a granularidade para marcação de tempo válido da classe C de **granularidade de** C.

Consideremos inicialmente o caso em que a granularidade de C é mais grossa que a de C_1. Neste caso, não haveria problemas de compatibilização entre C e C_1, pois uma instância de C_1 poderia substituir uma instância da classe C em qualquer situação, seguindo o paradigma OO. Qualquer operação que utilize valores de tempo com a granularidade da classe C poderá ser realizada com valores da granularidade da classe C_1 (mais precisos). Consideremos, por exemplo, que a granularidade de C seja de anos e a granularidade de C_1 de meses. Uma operação que atualize uma instância

de C especificando um tempo de validade expresso em anos poderá ser executada diretamente sobre uma instância de C_1, onde o tempo de validade será automaticamente convertido à granularidade adequada. Uma consulta expressa com a granularidade de C aplicada sobre instâncias de C_1, terá uma resposta expressa na granularidade de C_1 o que não causa problemas.

Quando a superclasse C for definida com granularidade mais fina que a subclasse C_1, teremos que compatibilizar a classe C_1 com sua superclasse, pois nem todas as operações aplicáveis sobre objetos de C serão possíveis em instâncias de C_1. Por exemplo, considerando-se agora a classe C definida com granularidade de meses e a classe C_1 com granularidade de anos, o que aconteceria se atribuíssemos um valor a um objeto pertencente à classe C_1 como tendo validade em um único mês de um ano? Como C_1 não pode expressar esta granularidade teríamos que considerar o valor válido por todo o ano, ou por nenhum instante dele. O processo de temporalização deve, então, compatibilizar as granularidades de classes com suporte a tempo válido ligadas por laços de herança. A granularidade das subclasses deve ser, no mínimo, tão precisa quanto a granularidade de suas superclasses. A temporalização por herança com respeito à granularidade segue em linhas gerais a temporalização por herança quanto às suas categorias temporais (seção 3.2.1). Na primeira compatibilização por herança a nova subclasse com a granularidade requerida substitui a classe com a granularidade antiga, eliminando-a.

3.4.2 Composição

O tratamento para compatibilizar as granularidades das classes ligadas por composição é semelhante ao processo de compatibilização com respeito à variação de valores. Utilizaremos o exemplo da seção anterior para explicar o processo.

Exemplo:

Classe *REVISTA* **bitemporal discreta com granularidade mês**
TÍTULO : STRING;
EDITOR : PESSOA;
NÚMERO : INTEIRO;
PREÇO : DINHEIRO **discreto com granularidade mês.**

Classe *PESSOA* **de tempo válido escada com granularidade ano**
NOME : STRING;
PESO : INTEIRO **contínuo com granularidade ano.**

Classe *DINHEIRO* **de tempo de transação**
MOEDA : STRING;
VALOR : REAL.

□

De maneira similar ao caso de variação de valores, as granularidades das classes componentes podem seguir três casos com respeito a granularidade das classes que compõem:

1. Quando a classe componente com suporte a tempo válido é resultado da temporalização, sua granularidade seguirá, por *default*, a granularidade da classe que a originou. Um exemplo deste caso é a classe *STRING* bitemporal discreta originada pela especificação de *TÍTULO*, componente de *REVISTA*, que terá granularidade de mês.

2. Quando a classe componente já estava definida com suporte a tempo válido antes da temporalização e, portanto, já tinha uma granularidade definida, não se processará, por *default*, nenhuma alteração durante o processo de temporalização — mesmo quando as granularidades das classes composta e componente forem diferentes. Temos como exemplo deste caso a classe bitemporal *PESSOA* componente de *REVISTA*. A diferença entre as granularidades de *REVISTA* e *PESSOA* não causa problemas — embora os valores de objetos da classe *PESSOA* variem apenas anualmente, a ligação entre objetos de *PESSOA* com objetos da classe *REVISTA* poderá variar mensalmente.

3. Quando não desejamos que a classe componente assuma a granularidade determinada por *default* em um dos casos anteriores, devemos indicar a granularidade desejada na declaração da componente correspondente. É este o caso da classe *INTEIRO* de tempo válido contínuo com granularidade diária (de *PESO*). Como dito anteriormente, não há problemas quando a classe composta possuir granularidade diferente da classe componente. Contudo, quando a classe componente já estava definida com suporte a tempo válido antes da temporalização e especificamos uma nova granularidade para ela na declaração da classe composta, a granularidade especificada deve ser, ao menos, tão precisa quanto a original. Esta restrição é necessária para garantir que todas as classes originadas na temporalização por composição possam executar as operações temporais possíveis na classe primitiva.

A figura 3.2 representa o esquema original utilizado nos exemplos vistos até aqui. As figuras 3.3 e 3.4 representam o esquema resultante após a execução do processo completo de temporalização. Linhas contínuas representam ligações de composição, enquanto linhas tracejadas indicam ligações de herança.

3.5 Resumo da Temporalização

Por envolver as várias características temporais em que as classes do BD podem diferir, o processo de temporalização foi apresentado de forma extensa e gradual nas seções anteriores. Nesta seção resumimos o processo visando dar uma visão integral dele.

A função da temporalização é compatibilizar as classes do BD com respeito às suas características temporais — categoria temporal, tipo de variação de valores e granularidade. Como as classes estão relacionadas por herança e por composição, a temporalização deve ser executada em duas fases: compatibilizando as classes de acordo com suas relações de herança e, em seguida, de acordo com as relações de composição. O processo de compatibilização por herança/composição deve ser executado diversas vezes, até que não produza mais alterações no esquema. Apesar de ser dividido em duas fases, que poderão ser repetidas diversas vezes, o processo deve ser executado atomicamente.

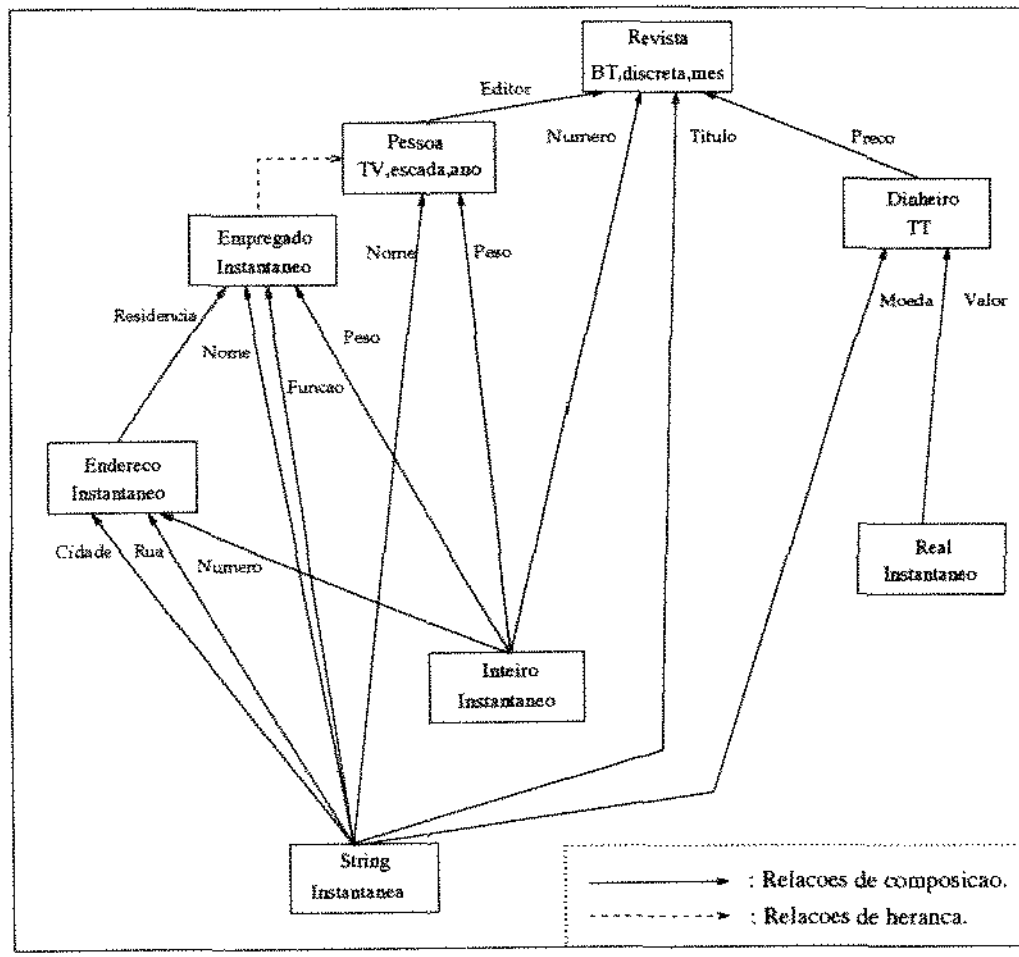

Figura 3.2: Esquema original.

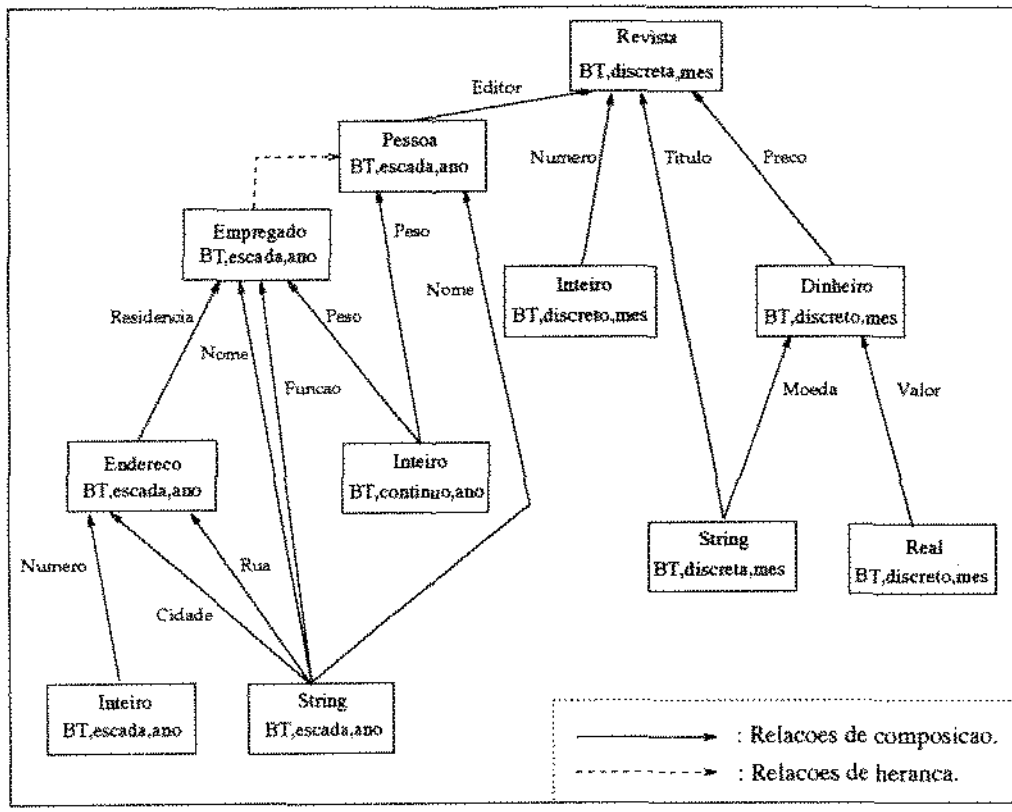

Figura 3.3: Esquema temporalizado — Parte 1.

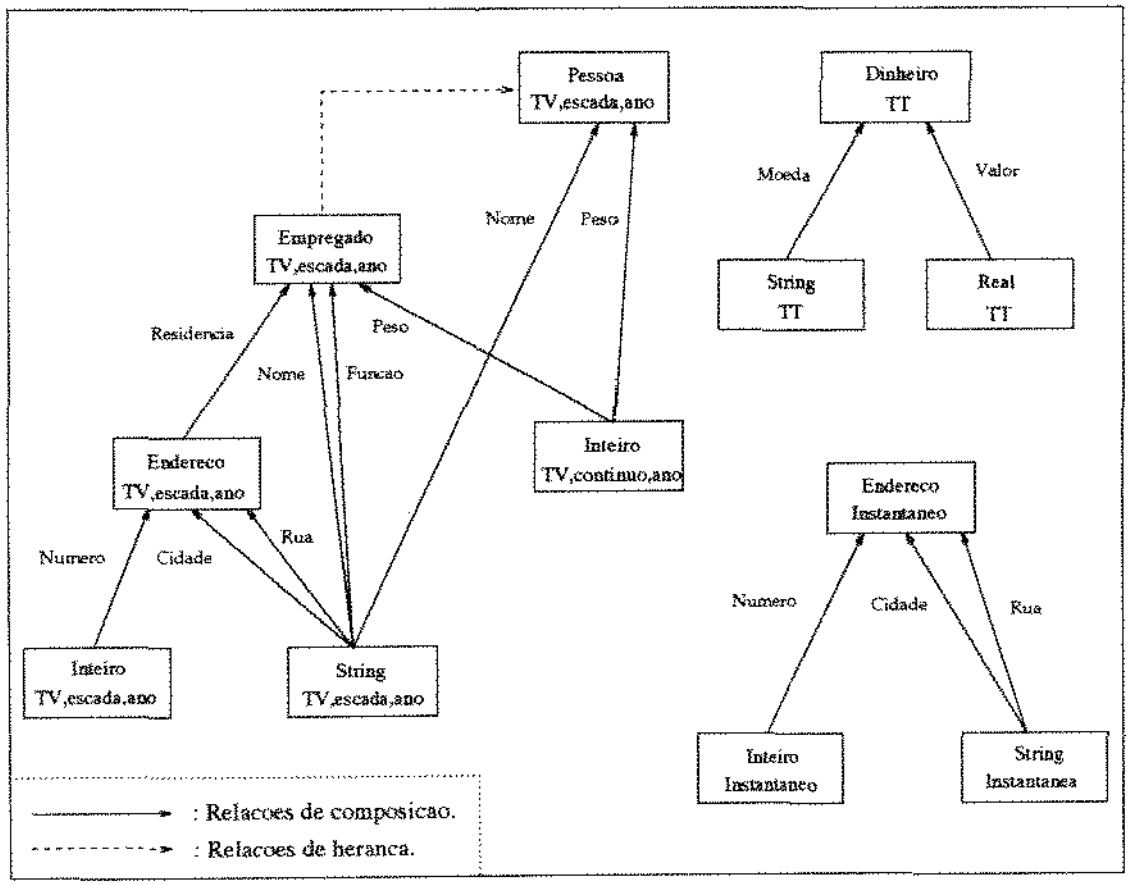

Figura 3.4: Esquema temporalizado Parte 2.

A temporalização altera as características temporais de algumas classes e cria outras classes com características diferentes a partir das classes originais. Portanto, após a temporalização, para cada classe original definida pelo usuário, poderemos ter um conjunto de classes que diferem apenas em suas características temporais. Denominaremos este conjunto de classes de **família de classes**, a classe definida pelo usuário de **classe primitiva** e as classes originadas na temporalização de **classes derivadas**.

A temporalização por **herança** é necessária para manter o paradigma da OO, segundo o qual um objeto de uma superclasse pode ser substituído por um objeto de sua subclasse em qualquer circunstância, o que resulta na necessidade de compatibilização das subclasses com suas superclasses. Quando uma subclasse precisa ter suas características temporais alteradas para poder executar as operações temporais possíveis em suas superclasses, a subclasse primitiva é eliminada, sendo substituída pela classe derivada. Isto acontece porque todos os objetos da subclasse devem poder substituir objetos da superclasse e, portanto, pertencer à classe derivada com as novas características temporais. A classe derivada será considerada então a própria classe primitiva.

Analisamos agora a necessidade da temporalização por herança para compatibilizar as classes primitivas pelas três características temporais das classes envolvidas.

Categoria Temporal: Necessita da compatibilização porque uma classe de uma categoria temporal só aceita as operações temporais possíveis em classes de sua categoria ou de categorias anteriores na ordem parcial.

Variação de Valores: Não precisa da compatibilização porque todas as operações temporais possíveis em objetos da superclasse continuariam possíveis em objetos de suas subclasses, mesmo quando as classes tiverem tipos de variação diferentes.

Granularidade: A compatibilização é necessária porque a subclasse deve ter ao menos a mesma precisão na marcação de tempo válido que a subclasse para poder substituí-la.

Caso uma classe derivada na temporalização por herança tenha suporte a tempo válido, mas a classe primitiva não, a granularidade e a variação de valores da classe derivada serão as mesmas da superclasse que a originou na temporalização.

A temporalização por **composição** se faz necessária porque, para recuperarmos um objeto composto em um certo tempo, temos que obter o conteúdo de todos os seus componentes neste tempo. No entanto, ao se criar a classe derivada na temporalização por composição, não se deve eliminar a classe primitiva, pois podem existir instâncias da classe primitiva que sejam componentes de outras classes (e, portanto, devem ser compatíveis com estas) ou que não sejam componentes de nenhuma classe. Esta fase da temporalização cria classes derivadas sem eliminar a classe primitiva.

Analisamos a seguir a necessidade de temporalização por composição das classes primitivas segundo as três características temporais das classes envolvidas.

Categoria Temporal: A compatibilização é necessária porque, se desejarmos conhecer o conteúdo de um objeto em um tempo (válido ou de transação) específico, seus componentes também precisarão ter suporte ao mesmo tipo de tempo para poderem ser recuperados.

Variação de Valores: Não é necessária a compatibilização das classes primitivas, pois a variação de valores da classe composta determina de que forma os objetos das classes componentes se ligam aos objetos compostos, enquanto a variação de valores da classe componente indica

como os valores dos objetos componentes variam. Portanto, as variações de valores das classes composta e componentes não precisam ser iguais. Contudo, se o usuário desejar definir para uma classe componente uma variação de valores diferente da definida na classe primitiva, a nova variação pode ser determinada ao se definir a classe como componente.

Granularidade: Não é necessário compatibilizar as classes primitivas, já que a granularidade da classe composta determina com que freqüência a ligação entre os objetos composto e componentes pode variar, enquanto a granularidade da componente determina a freqüência com que os valores desta classe variam. As classes composta e componente podem, portanto, ter granularidades diferentes. Contudo, se o usuário desejar definir para uma classe componente uma granularidade diferente da definida na classe primitiva, a granularidade desejada pode ser determinada ao se definir a classe como componente. Neste caso, a nova granularidade deve ser tão fina quanto a definida na classe primitiva, para que as classes derivadas sejam capazes de executar as operações temporais possíveis na classe primitiva.

Caso surjam classes derivadas com suporte a tempo válido, mas a classe primitiva não tenha suporte a este tipo de tempo, as classes derivadas terão, por *default*, a mesma granularidade e variação de valores das classes compostas que as originaram. Se o usuário desejar uma granularidade e/ou variação de valores diferente, poderá especificá-la(s) na declaração das variáveis de instância correspondentes.

Após o processo de compatibilização por herança/composição ser executado a primeira vez, as classes derivadas podem precisar ser compatibilizadas. Considere como exemplo duas classes C e C_1, C superclasse de C_1. Suponha que durante a compatibilização por composição surjam classes derivadas de C. Neste caso, deveriam existir classes da família de C_1 que fossem compatíveis com as novas classes da família de C. Portanto, uma nova compatibilização por herança se faz necessária. Note-se contudo que, na nova compatibilização poderiam ser criadas novas classes da família de C_1, mas as classes já existentes desta família não seriam afetadas.

O surgimento de novas classes causaria a necessidade de nova compatibilização por composição, que, por sua vez, poderia provocar nova compatibilização por herança, e assim sucessivamente. O processo de temporalização findará apenas quando as compatibilizações não provocarem mais alterações no esquema.

3.6 Famílias de Classes

Como dissemos na seção anterior, a temporalização de uma classe primitiva pode originar uma família de classes. Nesta seção descrevemos o tratamento do SGBD às famílias de classes.

Depois de terminado o processo de temporalização as classes de cada família tornam-se independentes umas das outras. Apenas em algumas situações especiais a ligação existente entre as classes de uma família será relevante. Na maioria das situações o usuário tratará as classes como não possuindo nenhuma relação devida à temporalização.

Quando na definição de uma classe C, o usuário referenciar uma outra classe de uma família qualquer apenas pelo seu nome, o SGBD considerará como alvo desta referência a classe primitiva desta família, caso a classe tenha sido definida pelo usuário, ou a classe original do sistema (instantânea), caso seja uma classe básica provida pelo SGBD.

Na criação de um objeto, o usuário deve determinar em que classe ele será criado através das relações de composição que ele possua. Consideremos três classes C_1 (de tempo de transação),

C_2 (de tempo válido, com variação escada e granularidade dia) e C_3 (instantânea), onde C_3 é componente de C_2 e esta é, por sua vez , componente de C_1. Com a temporalização surgirão as classes $C_{2/BT,escada,dia}$, $C_{3/TV,escada,dia}$ e $C_{3/BT,escada,dia}$. Na criação de um objeto O_3 na família de classes C_3, o usuário determinaria a classe exata na qual O_3 seria criado da seguinte forma:

- Caso O_3 seja um objeto independente, ou seja, não faça parte da composição de nenhum objeto de C_2, o usuário deverá criá-lo em $C_{3/instantânea}$.
- Caso O_3 seja criado em decorrência da criação de um objeto O_2, da família de classes C_2, e O_2 seja um objeto independente, O_3 deverá ser criado na classe $C_{3/TV,escada,dia}$.
- No caso da criação de um objeto O_1 na classe C_1, seria necessária a referência (ou criação) de um objeto, O_2, em C_2 e, conseqüentemente, um objeto O_3, componente de O_2, em C_3. Neste caso O_3 deverá ser criado na classe $C_{3/BT,escada,dia}$.

Normalmente, o número de classes derivadas para cada classe primitiva não é elevado, embora isto possa ocorrer. Nas seções anteriores descrevemos o processo de temporalização, considerando que as classes possam ter características temporais totalmente distintas, mesmo quando relacionadas por herança e/ou composição. Todavia, as classes pertencentes a cada aplicação, tendem a possuir características temporais semelhantes, o que diminuiria o número de classes derivadas, facilitando a sua manipulação pelo usuário.

3.7 Identidade de Objetos

Assim como [CC88, KC86], consideramos que a manutenção da identidade dos objetos torna-se ainda mais importante em um BDT. A identidade de um objeto (ou sua "essência" [CC88]) deve ser mantida com o passar do tempo, funcionando como o elo de ligação entre os diversos valores assumidos pelo objeto.

Algumas questões relacionadas à identidade de objetos devem ser estudadas ao considerarmos um BDT, a saber:

- A existência de objetos está associada a tempo válido ou tempo de transação?
- Qual o significado das operações de exclusão e inclusão em um BDT?
- Como se efetua a migração de objetos?

Analisamos estas questões a seguir.

3.7.1 Existência de Objetos

Consideremos o caso de um objeto R_1 incluído na classe *REVISTA* em janeiro de 1980. Ao incluirmos R_1, conhecemos, além de seus dados correntes, dados desde o ano de 1970. Adicionalmente, temos previsões para seus dados até 1990. A questão que se coloca então é: em que tempos a revista R_1 existiu? A resposta para esta questão é que o **objeto** R_1 só começou a existir a partir de janeiro de 1980, mas a **revista** R_1 existiu entre 1970 e 1990. Ou seja, mantém-se a função dos dois tipos de tempo — tempo de transação descreve a vida dos objetos enquanto tempo válido descreve a vida das entidades modeladas pelos objetos.

3.7.2 Exclusão e Inclusão de Objetos

A operação de exclusão de um objeto adquire uma nova interpretação em um BDT. Ao excluirmos um objeto, não eliminamos, necessariamente, todos os seus dados. A eliminação dos dados de um objeto, bem como de sua identidade (*oid*), só ocorreria quando o objeto excluído pertencesse a uma classe sem suporte a tempo de transação. Contudo, como veremos adiante, um objeto pode pertencer a classes de diferentes categorias temporais em momentos distintos (devido a migrações). Portanto, a exclusão de um objeto só resulta na eliminação de seu *oid* quando o objeto em questão não tiver pertencido em nenhum momento a uma classe de tempo de transação ou bitemporal.

A exclusão de um objeto pode ser vista como a retirada deste objeto do estado corrente dos dados. Desta forma, o objeto não mais acompanharia a evolução do BD. Caso o objeto tenha pertencido a uma classe com suporte a tempo de transação, seu *oid* continua existindo, associado a estados passados. Caso contrário, seria eliminado.

Com a nova interpretação para a operação de exclusão, a inclusão passa a possuir duas funções. A primeira é a criação de um novo objeto, gerando novo *oid* — que é a sua interpretação em um BD convencional. A segunda interpretação é a reativação de um objeto previamente excluído, i.e., a reassociação do objeto ao estado corrente do BD.

3.7.3 Migração de Objetos

É natural que, com o passar do tempo, objetos apresentem mudanças em suas características. Estas alterações podem provocar, além das modificações corriqueiras em seus valores, a migração de objetos de uma classe para outra (e.g., um objeto da classe *EMPREGADO* pode ter que migrar para uma classe *GERENTE*, quando a pessoa por ele representada for promovida, ou para uma classe *PESSOA*, no caso de uma demissão).

Ao ser migrado, um objeto perde sua relação de composição com os objetos dos quais faz parte. O motivo disto é que a classe de destino da migração pode não possuir as características (componentes, comportamento e características temporais) necessárias para que a relação de composição continue existindo. Por exemplo, se um objeto migra da classe *STRING* para a classe *PESSOA* não poderá continuar a ser componente de objetos que esperem um componente *STRING*. O objeto migrado não perderá suas ligações com os objetos que compõe apenas quando a classe de destino da migração possua características que sejam compatíveis com a manutenção desta relação. Isto ocorrerá em alguns casos de migração entre classes de uma mesma família e entre super e subclasses. Os objetos componentes do objeto migrado manterão a relação de composição com o objeto migrado, à exceção do caso em que a classe a que pertencem não fizer parte da classe destino.

Na migração de um objeto duas questões básicas precisam ser definidas, a saber, (1) o que acontecerá com os dados do objeto e (2) que dados serão transferidos para a nova classe (e como será a transferência).

Destino dos Dados Já Existentes

O que acontecerá com os dados do objeto a ser migrado dependerá da categoria temporal da classe à qual o objeto pertencia antes da migração (classe origem). Se a classe origem tiver suporte a tempo de transação (i.e., for uma classe bitemporal ou de tempo de transação) todos os seus dados permanecerão inalterados e acessíveis no BD. Caso a classe origem não tiver suporte a tempo de

transação (classe instantânea ou de tempo válido) seus dados serão perdidos, excetuando-se aqueles que forem passados à classe destino.

Aqui convém relembrar que o objetivo do suporte ao tempo de transação é justamente a manutenção de estados passados do BD — classes com suporte a tempo de transação mantêm a história do BD. Por isto, todos os dados relativos ao objeto migrado enquanto este pertencia à classe origem devem ser preservados. Classes sem suporte a tempo de transação representam apenas conhecimento atual (visão corrente dos dados). Portanto, não há sentido na manutenção da situação antiga dos dados do objeto migrado. Considere, por exemplo, um objeto O criado em uma classe bitemporal, que migre no tempo t_1 para uma classe instantânea e posteriormente migre novamente, no tempo t_2, para uma classe de tempo de transação. Entre o tempo de sua criação e t_1 seus dados ficam armazenados; entre t_1 e t_2 não há registro de sua existência; e após t_2, novamente seus dados passam a ser guardados. A situação de O entre t_1 e t_2 equivaleria a ele ter sido removido em t_1 e reincluído, desta vez em uma classe de tempo de transação, no tempo t_2.

Transferência de Dados entre as Classes Origem e Destino

As classes origem e destino da migração do objeto possuirão provavelmente algumas componentes em comum. Esta seção visa estabelecer como os dados destas componentes serão passados da classe origem à classe destino, conforme as características temporais de ambas as classes. Componentes pertencentes exclusivamente à classe destino terão seus valores preenchidos com nulos, que podem ser substituídos posteriormente pelo usuário.

Independente das classes origem e destino, os dados transferidos corresponderão sempre ao estado do BD imediatamente anterior ao instante da migração.

Caso a classe origem não ofereça suporte a tempo válido, os dados transferidos da classe origem serão passados à classe destino, como se resultassem de uma operação de reinclusão (mantendo o *oid*). Se a classe destino também não tiver suporte a tempo válido estes dados serão simplesmente copiados como correntes a partir do próximo estado do BD. Se a classe destino oferecer suporte a tempo válido, a validade dos dados transferidos é determinada pelo sistema de *defaults* do SGBD, de acordo com o tipo de variação de valores da classe destino. Caso a classe origem suporte tempo válido, mas a classe destino não, serão transferidos apenas os dados válidos em *now*.

O caso mais complexo ocorre quando as classes origem e destino possuem suporte a tempo válido. Neste caso, o histórico da classe origem é copiado para a classe destino. A complexidade se dá porque as características temporais das duas classes podem ser diferentes. Nos casos anteriores o tipo de variação de valores e a granularidade das classes não eram relevantes porque sempre se passava um único valor, que era inserido de forma convencional na classe destino.

Descrevemos inicialmente o tratamento da variação de valores e em seguida tratamos da questão da granularidade.

O critério para transferir o histórico da classe origem para a classe destino é bastante simples: cada ponto de tempo válido deverá ter o mesmo valor nos históricos das classes origem e destino. Se ambas as classes tiverem o mesmo tipo de variação, os históricos serão iguais. Quando a classe origem for discreta ou escada os valores devem ser copiados para os pontos de tempo equivalentes na classe destino e os demais instantes devem ser preenchidos com nulos (se a classe destino for contínua, por exemplo, estes nulos devem impedir que sejam inferidos valores para estes pontos de tempo, baseando-se em outros pontos de tempo). Quando a classe origem for contínua (ou definida pelo usuário), devem ser calculados e transferidos os valores relativos a cada ponto de tempo da

granularidade da classe destino.

Caso a granularidade das classes origem e destino sejam idênticas, o processo de transferência explicado acima não sofre alterações. Quando a granularidade da classe origem for mais grossa que a granularidade da classe destino opera-se a conversão normal de granularidades (veja seção 3.4) antes da transferência dos dados, e a transferência ocorre segundo o procedimento normal. Quando a granularidade da classe origem for mais fina que a da classe destino é preciso estabelecer um critério para converter os dados de uma granularidade para outra. Podemos estabelecer que os valores válidos pela maior parte do período de granularidade mais grossa seriam considerados como válidos durante todo o período (e.g., numa transferência entre uma classe de granularidade mês e outra de granularidade ano, os dados passariam a classe destino como válidos em um ano se, na classe de origem, fossem válidos por mais de seis meses deste ano). Outras opções seriam considerar os valores válidos no primeiro instante do período como válidos para todo o período (os valores seriam considerados válidos em um ano na classe destino caso fossem válidos no mês de janeiro na origem) ou considerar válidos para um ponto de tempo na classe destino apenas os valores que foram válidos durante todo o período correspondente na classe origem (os dados seriam considerados válidos para um ano na classe destino caso fossem válidos de janeiro a dezembro deste ano). Estabelecido o critério, a transferência seguiria o esquema normal.

3.8 Esquemas

A possibilidade de evolução de esquema aliada a manutenção de estados passados do BD origina alguns novos problemas a serem estudados. Nesta seção descrevemos como são armazenados esquemas de estados passados e tratamos as questões de herança e composição temporais e evolução de esquema.

3.8.1 Esquemas Passados

Como dissemos na seção 3.1 esquemas de estados passados contêm apenas classes bitemporais e de tempo de transação, pois são as únicas que mantêm dados passados do BD. Todavia, devemos lembrar que as classes possuem ligações entre si e as definições das classes mantidas em esquemas passados não podem ser prejudicadas pela não manutenção das classes sem suporte a tempo de transação.

As relações de composição não afetam a correção da definição de esquemas passados. Sejam CT, uma classe com suporte a tempo de transação, e C uma classe sem suporte a este tipo de tempo. Caso CT seja componente de C em algum esquema passado, a não manutenção de C neste esquema não afetaria a definição de CT. O caso inverso, C componente de CT, não poderia ocorrer, devido ao processo de temporalização.

As relações de herança entre classes com e sem suporte a tempo de transação afetam as definições de classes de esquemas passados. Sejam C e CT como definidas acima. Pelo processo de temporalização asseguramos que CT não pode ser superclasse de C em qualquer dos esquemas passados. Entretanto, caso C seja superclasse de CT haverá um problema, pois embora C não seja preservada em esquemas passados, as características de C herdadas por CT devem ser preservadas. Para solucionar esta questão, estabelecemos que sempre que uma classe com suporte a tempo de transação for subclasse de classe(s) instantânea(s) ou de tempo válido, essa classe será armazenada,

em esquemas referentes a estados passados, com a definição de todos os métodos e componentes herdados diretamente destas superclasses.

3.8.2 Herança e Composição Temporais

A manutenção de diversos esquemas possibilita um novo tipo de herança. Ao invés de definirmos que uma classe C herda as características de uma classe K, podemos dizer que C herda as características de K, como K estava definida em um tempo de transação tt_1 — obviamente a classe K deve estar definida no esquema referente a tt_1. Denominamos este tipo de herança de herança temporal.

Como não podemos ligar os grafos de herança atual e do tempo tt_1, pois isto contrariaria a definição de BDT, que considera cada estado independente dos demais, nem podemos incluir K no esquema atual, já que K pode estar definida de forma diferente no estado corrente, ficamos entre duas alternativas, a saber:

- a criação de um tipo especial de classe, que não poderia possuir instâncias, utilizada apenas para a herança temporal; ou
- a definição das características herdadas diretamente na subclasse em questão.

Optamos pela segunda alternativa, pela sua maior simplicidade e pela semelhança com a solução adotada para o armazenamento de esquemas passados. Note que as características temporais da subclasse também poderiam ser modificadas devido ao processo de temporalização por herança. Além disto, os métodos e componentes herdados teriam que ser assinalados de alguma forma, para o caso do usuário desejar desfazer a ligação de herança temporal futuramente.

À primeira vista poderíamos imaginar um tipo de composição temporal, que seria análoga à herança temporal. Ou seja, poderíamos definir uma classe C como tendo como componente uma classe K, como esta era definida em um tempo de transação tt_1. No entanto, neste caso a classe K não seria utilizada apenas como referência para definição das características de uma outra classe. A classe K precisaria conter instâncias que serviriam de componentes de objetos da classe C. Portanto, K deveria ser uma classe real no estado corrente do BD, e a composição tornar-se-ia uma composição convencional.

3.8.3 Evolução de Esquema

Mudanças no esquema são muitas vezes inevitáveis. Em nosso modelo estas mudanças são permitidas pela associação de cada esquema aos períodos de tempo de transação em que ele foi ativo. A questão de evolução de esquema tem sido extensivamente estudada na literatura (e.g., [BKKK87, KC88, PS87, NR89]). Estes artigos procuram, entre outras coisas, definir maneiras de implementar a evolução de esquema em sistemas específicos (estruturas de dados e algoritmos) e, principalmente, garantir que o novo esquema seja "consistente" — não contrarie nenhuma característica do modelo e corresponda às expectativas que o usuário tinha ao executar as alterações. Nesta seção, não cuidamos destes aspectos da evolução de esquema. Restringimos nossa análise às conseqüências da evolução de esquema sobre as características temporais de nosso modelo.

Se, em uma mudança no esquema, uma classe instantânea ou de tempo válido sofrer qualquer modificação, ou mesmo for removida, a definição antiga da classe será perdida. Classes destas categorias temporais, por não suportarem tempo de transação, estão associadas apenas ao último estado do BD e, logo, não constarão de esquemas relativos a estados passados do BD. Os dados

referentes à antiga definição da classe serão perdidos. Classes com suporte a tempo de transação, ao contrário, têm todos os seus dados e suas definições mantidos inalterados após qualquer mudança no esquema. Estes dados estarão associados ao esquema relativo ao tempo de transação anterior à mudança no esquema.

Além do destino das informações do esquema, precisaremos definir como ficará o relacionamento entre as classes após a mudança e como os dados do esquema antigo passarão ao novo esquema. Para explicar estas questões, classificamos as mudanças de esquema nos seguintes grupos:

- Criação / Remoção de uma Classe;
- Mudanças no Grafo de Composição (eliminação, adição ou mudança no tipo de componentes);
- Mudanças no Grafo de Herança (criação ou eliminação de arestas); e
- Mudanças nas Características Temporais de uma Classe.

3.8.4 Criação / Remoção de uma Classe

Analisamos aqui apenas os efeitos sobre as classes criadas/removidas. Os efeitos em outras classes são cobertos nos outros tipos de mudanças no esquema (e.g., o efeito da eliminação de uma superclasse de uma classe qualquer é equivalente nesta classe à eliminação da relação de herança entre estas classes).

A criação de uma classe não traz nenhum efeito inesperado. Ao ser criada uma classe, o único cuidado a ser tomado é a sua compatibilização com o restante do esquema, através da temporalização.

Se for permitida a remoção de classes não vazias, a remoção de uma classe implicará na remoção de todas as suas instâncias a partir do tempo de transação da operação. Como dissemos anteriormente, se a classe removida tiver suporte a tempo de transação, sua definição continuará fazendo parte do esquema do BD nos estados entre sua criação e sua remoção. Seus dados também permanecem inalterados no BD, ligados a estes estados passados. Caso a classe removida seja instantânea ou de tempo válido, não será mantido nenhum registro de seus objetos, nem da própria classe.

Como as classes de uma família tornam-se independentes após a temporalização, a remoção de uma classe qualquer de uma família (mesmo a primitiva) não afeta as demais.

3.8.5 Mudanças no Grafo de Composição

A adição de uma classe C_1 como componente de outra classe C faz com que a classe C_1 (e suas classes componentes sucessivamente) tenha que ser compatibilizada com a classe C. Por este motivo, o processo de temporalização deve ser reexecutado e a família de classes a que pertence C_1 (bem como outras famílias) pode ser aumentada. As instâncias da família de classes C_1 não são afetadas com a mudança, pois nenhuma das classes já existentes da família C_1 seria alterada — poderiam apenas surgir novas classes na família.

A remoção da relação de composição de uma componente C_1 com uma classe C não traz efeitos colaterais. As classes C e C_1 continuam a existir independentemente, mas seus objetos perdem as ligações de composição entre eles. Se o usuário interpretar que a existência de C_1 dependia exclusivamente de sua composição com C, deve efetuar a remoção de C_1 manualmente.

Finalmente, as conseqüências de uma mudança no domínio de uma componente seriam as mesmas da remoção e/ou criação de uma componente. A mudança no nome de uma variável de instância não provocaria nenhuma conseqüência sobre os aspectos temporais do modelo.

3.8.6 Mudanças no Grafo de Herança

Dois casos devem ser considerados: a transformação de uma classe em subclasse de outra e o caso inverso, a eliminação do relacionamento de herança entre duas classes.

Se uma classe C_1 for transformada em subclasse de outra classe C, C_1 deverá ser temporalizada para se compatibilizar com a classe C. No entanto, a primeira temporalização por herança causa a eliminação da classe alterada, substituindo-a pela nova classe com as características temporais adequadas. Portanto, a operação equivaleria, no caso de C_1 sofrer alguma alteração, a mudança das características temporais de uma classe, que será explicada na seção seguinte. As classes que porventura sejam derivadas de C_1 serão incorporadas à família de C_1.

A eliminação dos laços de herança entre duas classes C e C_1 não produz efeitos sobre as características temporais do modelo. C e C_1 continuam a existir de forma independente. Caso C_1 tenha se originado pela ligação de herança entre as classes primitivas das famílias de C e C_1, e o usuário interprete que a existência de C_1 dependia de sua ligação com C, ele deve remover a classe C_1 explicitamente.

Devem ser considerados também os efeitos sobre as classes componentes herdadas. A alteração no grafo de herança pode provocar a adição ou remoção de classes componentes, cujas conseqüências foram vistas na seção anterior.

3.8.7 Mudanças nas Características Temporais de uma Classe

Este tipo de mudança é o que pode provocar maiores conseqüências no BD. Estudaremos primeiramente os efeitos devidos às relações de herança, passando em seguida aos efeitos devidos às relações de composição.

O primeiro fator a ser analisado é relativo à própria classe cujas características temporais foram alteradas. Denominaremos esta classe de C. C deve ser compatibilizada com suas superclasses, portanto, as novas características de C poderão ser alteradas novamente durante a temporalização por herança. A alteração pode, inclusive, ser anulada durante a temporalização (e.g., se mudamos a categoria temporal de C de bitemporal para instantânea, mas existe uma superclasse de C que é bitemporal, na temporalização C voltaria a ser bitemporal). O destino das instâncias de C será tratado mais à frente, quando discutirmos a questão da composição.

Com a mudança de C, suas subclasses devem ser novamente compatibilizadas com suas características. O efeito será equivalente à alteração das características temporais das subclasses.

Após a alteração das características temporais de uma classe por herança, o esquema deve ser retemporalizado também no que diz respeito à composição. Este processo de temporalização é executado da mesma forma que após a definição inicial do esquema, quando todas as classes estavam vazias. Devemos então decidir o destino das instâncias das classes afetadas.

Dentro de uma família de classes os objetos serão migrados do esquema antigo para o novo, de forma a manter a função que desempenhavam no esquema original (e.g., se um objeto não for componente de outro objeto, deverá ser migrado para a classe correspondente à primitiva no novo esquema). Para melhor esclarecer os critérios para migração, utilizaremos o seguinte exemplo.

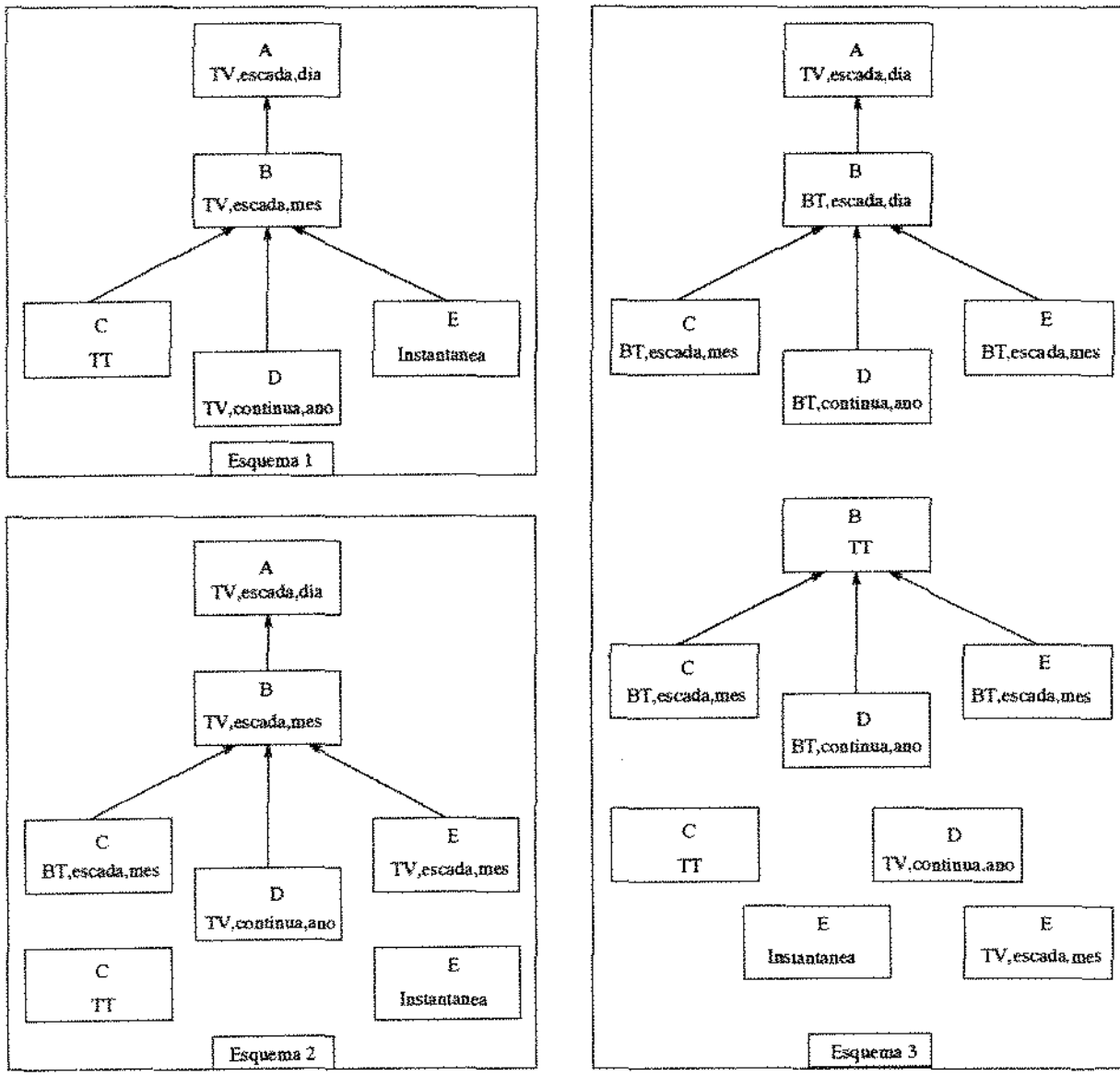

Figura 3.5: Evolução de esquema após alteração da classe B.

Exemplo:

Na figura 3.5 vemos três esquemas, contendo as classes A, B, C, D e E. O primeiro esquema corresponde à definição original do usuário. O segundo esquema resulta da temporalização do primeiro. O terceiro esquema é resultante do segundo esquema após a alteração da categoria de B de tempo válido para de tempo de transação. As ligações entre as classes correspondem a laços de composição. Para facilitar a visualização algumas classes são repetidas mais de uma vez em um mesmo esquema (e.g., $C'_{BT,escada,mês}$).

Descrevemos abaixo como será feita a migração dos objetos das classes envolvidas, do esquema 2 para o esquema 3. Na classe A não haverá migrações.

Família de Classes B: Instâncias componentes de A migrarão de $B_{TV,escada,mês}$ para $B_{BT,escada,dia}$ (que foi criada por B ser componente de A). Instâncias de B independentes (não componentes de outras classes) serão migradas de $B_{TV,escada,mês}$ para B_{TT}.

Família de Classes C: Instâncias independentes permanecerão na classe C_{TT}. Instâncias componentes de objetos da família de classes B permanecerão na classe $C_{BT,escada,mês}$.

Família de Classes D: Instâncias independentes permanecerão na classe $D_{TV,contínua,ano}$. Instâncias componentes de objetos da família de classes B migrarão para a classe $D_{BT,contínua,ano}$.

Família de Classes E: Instâncias independentes de E permanecerão na classe $E_{Instantânea}$. Instâncias componentes de objetos da família de classes B migrarão de $E_{TV,escada,mês}$ para $E_{BT,escada,mês}$.

□

Capítulo 4

Linguagem de Consulta TOOL

Este capítulo apresenta a linguagem de consulta TOOL (*Temporal Object-Oriented Language*) para o modelo TOODM, descrito no capítulo anterior. Escolhemos desenvolver a TOOL a partir da linguagem de consulta do modelo O_2, O2Query, ao invés de desenvolver uma linguagem completamente nova. Optamos por esta estratégia para facilitar o trabalho de especificação da linguagem, concentrando nossos esforços apenas nas construções temporais da TOOL. A linguagem O2Query foi escolhida por termos maior familiaridade com o sistema O_2, e pela disponibilidade deste sistema para uma futura implementação do TOODM e da TOOL.

Entretanto, TOOL não é específica apenas para o sistema O_2, assim como o TOODM não é somente uma extensão do modelo O_2. TOOL foi projetada tendo em vista o modelo OO descrito no capítulo 3 e, portanto, poderia ser adotada por qualquer sistema que possua características semelhantes.

O capítulo está organizado da seguinte forma. Na seção seguinte descrevemos resumidamente algumas características da linguagem O2Query, essenciais para o entendimento da TOOL. Na seção 4.2 iniciamos a descrição da TOOL introduzindo as construções da linguagem destinadas a manipular valores de tempo. Nas seções 4.3, 4.4, 4.5 e 4.6 descrevemos as principais construções da TOOL, destinadas a realizar as diversas consultas temporais. Os *defaults* da linguagem são descritos na seção 4.7. Por fim a seção 4.8 traz considerações finais sobre a linguagem.

4.1 Linguagem O2Query

O2Query é uma linguagem funcional definida por um conjunto de consultas base (*base queries*) que podem ser combinadas progressivamente para construir novas consultas.

O2Query é capaz de acessar a estrutura dos objetos, violando o encapsulamento, bem como de utilizar os métodos definidos para os objetos. O resultado de uma consulta pode ser um objeto já existente ou um **valor complexo** cujo tipo é definido na consulta. O resultado da consulta pode ser usado para construir novos objetos complexos e manipulado posteriormente.

O conceito de valores complexos é derivado do modelo de dados O_2 [LRV88]. Valores são semelhantes a objetos, contudo não possuem identidade, não são encapsulados e são manipulados por operadores primitivos, ao invés de métodos.

A linguagem O2Query apresenta uma grande variedade de consultas base. A TOOL estende a O2Query pela inclusão de algumas novas consultas base e a extensão das já existentes, intro-

duzindo novas cláusulas, para possibilitar as operações temporais. Descrevemos abaixo algumas características da O2Query (para maiores detalhes sobre O2Query veja [BCD89, GIP89]).

As consultas que servem como elementos base para o processo recursivo da linguagem são átomos ou nomes de entidades do BD (tais como classes).

Exemplos:

8 — *Consulta que retorna o valor 8.*
PESSOA — *Consulta que retorna os objetos da classe PESSOA.*

□

Além destas, há consultas base que têm como argumento listas, conjuntos, tuplas e valores atômicos. Abaixo enumeramos as consultas base sobre:

- valores numéricos: +, *, / e −.
- valores booleanos: AND, OR e NOT.
- tuplas: extração de campos por meio de um ponto.
- conjuntos: ELEMENT, INTER, MINUS, UNION, FLATTEN, COUNT, SUM e AVG.
- listas: ITH, HEAD, TAIL, SUBLIST, CONCAT, FLATTEN, COUNT, SUM e AVG.
- objetos: envio de mensagens.

Exemplos:

$2 + 2$ — *retorna o átomo 4.*
x AND y — *x e y são consultas que retornam valores booleanos; retorna um valor booleano.*
x.Nome — *x é uma consulta que retorna uma tupla; retorna o campo Nome de x.*
x INTER y — *x e y são consultas que retornam conjuntos; retorna o conjunto interseção dos conjuntos.*
FIRST (x) — *x é uma consulta que retorna uma lista; retorna o primeiro elemento de x.*
AVG (x) — *x é uma consulta que retorna um conjunto ou lista; retorna a média aritmética dos elementos de x.*
x $M(x_1,x_2)$ — *envia o método M ao objeto x com os argumentos* x_1 *e* x_2.

□

Valores complexos podem ser obtidos com os construtores TUPLE, SET e LIST.

Exemplos:

TUPLE (Nome: "Mário", Peso: 70)	*— retorna uma tupla com os campos Nome e Peso.*
SET (x_1,x_2)	*— retorna uma conjunto contendo os objetos x_1 e x_2.*
LIST (x_1,x_2,x_3)	*— retorna uma lista com os objetos x_1, x_2 e x_3.*

□

Para formação de predicados podem ser usados os comparadores =, <, >, <=, >= e <>, além dos quantificadores existencial e universal.

Exemplos:

a > b	*— a e b correspondem a valores numéricos; retorna um valor boolcano.*
FOR ALL x in y : p(x)	*— y é uma consulta que retorna um conjunto ou lista e p(x) é um predicado envolvendo a variável x: retorna verdadeiro se p(x) for verdadeiro para todos os elementos de y.*
EXISTS x in y : p(x)	*— y é uma consulta que retorna um conjunto ou lista e p(x) é um predicado envolvendo a variável x; retorna verdadeiro se p(x) for verdadeiro para pelo menos um elemento de y.*

□

Uma das consultas mais importantes da linguagem O2Query é o filtro para listas e conjuntos. Esta consulta base tem uma sintaxe SQL-like:

SELECT	q		q — consulta que pode envolver $x_1 \ldots x_n$.
FROM	$x_1 in f_1$		f_i — consulta que retorna um conjunto ou lista; pode envolver
	$\vdots$	$\Rightarrow$	um subconjunto das variáveis $x_1 \ldots x_{i-1}$.
	$x_n in f_n$		
WHERE	p		p — predicado que pode envolver $x_1 \ldots x_n$.

Esta consulta retorna um conjunto / lista consistindo do resultado de q sobre os elementos dos resultados das consultas f_i que satisfazem p.

Exemplo:

SELECT r.Editor
FROM r in Revista *Retorna os editores das revistas com preço > 100.*
WHERE r.Preço < 100

□

Por fim, qualquer consulta em O2Query receber um nome, através da operação DEFINE *nome* AS *consulta*, para ser reutilizada posteriormente.

4.2 Operações sobre Valores de Tempo em TOOL

Como dissemos no capítulo 3, nosso modelo lida com quatro tipos de valores de tempo, a saber, pontos de tempo, *spans*, intervalos e elementos temporais. Nesta seção descrevemos os operadores da TOOL para lidar com valores de tempo.

Listamos inicialmente as novas consultas base que têm valores de tempo como argumentos e retornam valores booleanos (comparadores), podendo, portanto, ser utilizadas para construção de predicados. Pontos, intervalos e elementos temporais podem ser combinados livremente como argumentos destes comparadores.

- BEFORE (arg_1,arg_2): verdadeiro quando o último instante de arg_1 é anterior ao primeiro instante de arg_2.
- AFTER (arg_1,arg_2): verdadeiro quando o instante inicial de arg_1 é posterior ao instante final de arg_2.
- OVERLAPS (arg_1,arg_2): verdadeiro quando os argumentos contêm ao menos um instante em comum.
- CONTAINS (arg_1,arg_2): verdadeiro quando arg_1 contém todos os instantes de arg_2.
- IN (arg_1,arg_2): verdadeiro quando arg_2 contém todos os instantes de arg_1.
- EQUAL (arg_1,arg_2): verdadeiro quando os termos contêm os mesmos instantes.
- MEETS (arg_1,arg_2): verdadeiro quando arg_2 inicia no instante seguinte ao término de arg_1.
- FOLLOWS (arg_1,arg_2): verdadeiro quando arg_1 inicia no instante seguinte ao término de arg_2.
- ADJACENT (arg_1,arg_2): verdadeiro quando arg_1 inicia no instante seguinte ao término de arg_2 ou vice-versa.

Relacionamos abaixo uma série de outras consultas base de uso geral para manipulação de valores de tempo. p_1 e p_2 representam pontos de tempo; i_1, i_2 e i_n representam intervalos e n indica um número inteiro.

• INTERVAL (p_1,p_2):	retorna um intervalo iniciado no instante p_1 e encerrado no instante p_2.
• TEMPORAL_EL (i_1,i_2,...,i_n):	retorna um elemento temporal composto pelos intervalos i_1,i_2,...e i_n.
• T_UNION (arg_1,arg_2):	aplicado sobre intervalos e/ou elementos temporais, retorna o elemento temporal resultante da união dos argumentos.
• T_MINUS (arg_1,arg_2):	aplicado sobre intervalos e/ou elementos temporais, retorna o elemento temporal resultante da diferença de conjuntos entre os argumentos.
• T_INTER (arg_1,arg_2):	aplicado sobre intervalos e/ou elementos temporais, retorna o elemento temporal resultante da interseção dos argumentos.
• FIRST_INSTANT / LAST_INSTANT / NTH_INSTANT (arg_1,[n]):	retorna o primeiro/último/n-ésimo instante de arg_1 (intervalo ou elemento temporal) – o segundo argumento só deve ser especificado com o operador NTH_INSTANT. A granularidade para o cálculo do instante a ser retornado será a mesma de arg_1.
• FIRST_INTERVAL / LAST_INTERVAL / NTH_INTERVAL (arg_1,[n]):	retorna o primeiro/último/n-ésimo intervalo do elemento temporal arg_1 – o segundo argumento só deve ser especificado com o operador NTH_INTERVAL.
• $s_1 + s_2$:	retorna o *span* resultante da soma de s_1 e s_2.
• $s_1 - s_2$:	retorna o *span* resultante da subtração de s_1 por s_2.
• s_1 * n:	retorna o *span* resultante da multiplicação da duração de s_1 por n.
• s_1 / n:	retorna o *span* resultante da divisão da duração de s_1 por n.
• arg_1 / s_1:	retorna o número de períodos de tempo de duração s_1 (dias, semanas, bimestres, períodos compostos de anos, meses, dias, etc) em arg_1 – um elemento temporal, intervalo ou outro *span*.

Um *span* representa uma quantidade de tempo, enquanto um ponto de tempo representa um ponto (intervalo) específico na linha de tempo. Por exemplo, o valor 1990 quando interpretado como um *span* representa 1990 anos. Quando interpretado como um ponto de tempo, porém, o valor representará o ano de 1990. Desta forma os operadores de soma, subtração, multiplicação e divisão são mais adequados a *spans* que a pontos de tempo. Consideramos que a estratégia para a execução destas operações é um detalhe relativo a implementação destes operadores e, portanto, não tratamos deste assunto.

4.2.1 Granularidade

O modelo dá ao usuário liberdade para escolher a granularidade mais conveniente às suas aplicações tanto em relação a tempo de usuário, quanto a tempo válido. Contudo, para medidas relacionadas a tempo de transação, controladas totalmente pelo SGBD, a granularidade utilizada é fixa.

Os operadores descritos na seção anterior lidam com valores de diferentes granularidades. A regra para trabalhar com operandos com precisões diferentes é converter o operando menos preciso para a granularidade do operando mais preciso. Esta conversão transforma cada valor de menor precisão no conjunto de todos os instantes nele contido, expressos numa precisão maior. Por exemplo, um valor com precisão a nível de mês, se convertido a uma granularidade de dias, englobaria todos os dias do mês.

Exemplos:

CONTAINS (%1990%, INTERVAL (%1990/03/01%, %1990/07/25%)) — *resulta em um valor verdadeiro, pois o primeiro argumento é convertido para o intervalo (%1990/01/01%, %1990/12/31%).*

T_INTER (%1993%, INTERVAL (%1992%, %1993/03%)) — *resulta no intervalo (%1993/01%, %1993/03%).*

□

Convencionamos delimitar valores de tempo com o símbolo "%". Os valores de tempo são especificados na forma *ano/mês/dia/hora/.../milissegundos*, sendo especificadas apenas as partes até a granularidade do valor.

Por fim, é necessário um novo operador que realize conversões de granularidades em valores de tempo operador este cuja estratégia de conversão possa ser escolhida pelo próprio usuário.

4.3 Consultas que Retornam Valores de Tempo

Com a manutenção de informações temporais no BD, novos tipos consultas tornam-se possíveis. Pode-se, por exemplo, determinar em que períodos de tempo certas condições dos dados foram verdadeiras, recuperar valores como conhecidos no passado ou selecionar um objeto de acordo com seu conteúdo no mundo real no passado ou futuro.

Nesta seção descrevemos os novos tipos de consultas incorporadas a O2Query para se obter acesso às informações temporais do BD. Explicamos as características de cada consulta de maneira informal, utilizando para isto vários exemplos. Utilizamos o esquema definido no capítulo anterior (páginas 63 e 64) como base para nossos exemplos.

Consultas em TOOL podem ser de duas categorias:

- Consultas que retornam valores de tempo; e
- Consultas que retornam objetos/estados temporais do BD.

Nesta seção apresentamos a primeira categoria de consultas e nas seções seguintes apresentamos a segunda classe de consultas em TOOL.

Definimos uma **expressão temporal** como qualquer consulta que retorne um ponto de tempo, intervalo ou elemento temporal — note que consultas que retornem *spans* não são consideradas expressões temporais. São portanto expressões temporais:

Exemplos:

%1982/12/25/10/30%	— *Ponto de tempo indicando 10h30min do dia 25 de dezembro de 1982.*
T_MINUS (%1980%, %1980/05%)	— *Retorna um elemento temporal contendo os intervalos (%1980/01%, %1980/04%) e (%1980/06%, %1980/12%).*
LAST_INSTANT (INTERVAL (%1972/02%, %1990%))	— *Retorna o ponto de tempo %1990/12%.*

□

Um tipo especial de expressão temporal é uma consulta aos dados do BD que retorne o tempo em que certas condições foram/serão verdadeiras. Este tipo de operação é permitida em alguns dos sistemas analisados no capítulo 2, como os de Gadia [Gad88] — operador WHEN (ω) — e de Clifford [CT85] — operador Ω. Duas novas consultas base foram incluídas na O2Query para realizar este tipo de operação, a saber, TWHEN e VWHEN.

4.3.1 TWHEN

A operação TWHEN retorna os valores de tempo de transação em que certos fatos estavam armazenados no BD. A sintaxe de uma consulta TWHEN é a seguinte:

```
TWHEN   predicado1
                                  AT
[VALID  [SOMETIME]  {  DURING  }  {  expressão_temporal1  } ]
                                  {  predicado2           }
                       AFTER
[FROM   conjuntos/listas/classes fonte                    ]
```

Utilizamos "[" e "]" para cláusulas facultativas e "{" e "}" quando uma das opções deve ser escolhida. Esta consulta retorna o conjunto de instantes de tempo de transação em que $predicado_1$ foi verdadeiro durante o tempo válido especificado na cláusula VALID. Se o $predicado_1$ envolver apenas dados com suporte a tempo de transação a cláusula VALID não deve ser especificada. Os instantes de tempo consecutivos são agrupados formando intervalos e o resultado final é expresso na granularidade mais grossa possível.

Exemplos:

4.3.1.1. Quando objetos de classe DINHEIRO de tempo de transação expressos em libras esterlinas estiveram presentes no BD?

TWHEN *d.Moeda = "Libra"*
FROM *d in Dinheiro*

Comentários:

- Como não foi especificada a qual classe da família *DINHEIRO* a consulta se refere, utiliza-se a classe primitiva da família *DINHEIRO* de tempo de transação.
- Como a consulta é executada sobre uma classe de tempo de transação, a cláusula VALID não deve ser especificada.
- A consulta retorna o elemento temporal, intervalo ou ponto de tempo contendo todos os instantes de tempo de transação em que existiu ao menos um objeto da classe *DINHEIRO* de tempo de transação cujo componente *MOEDA* tivesse o valor "Libra".

4.3.1.2. Quando o BD registrou que o empregado Marco residia em Curitiba em maio de 92?

TWHEN *e.Nome = "Marco" and e.Endereço.Cidade = "Curitiba"*
VALID AT *%1992/05%*
FROM *e in EMPREGADO(BT,escada,ano)*

Comentários:

- Utilizamos a notação *classe(categoria,variação,granularidade)* para indicar uma classe específica de uma família.
- A consulta pressupõe que Marco está na classe *EMPREGADO(BT,escada,ano)*; esta consulta não teria sentido se aplicada sobre a classe *EMPREGADO(TV,escada,ano)*, pois esta última não tem suporte a tempo de transação.
- Embora a granularidade da classe fonte seja de anos, a consulta pode ser expressa em qualquer granularidade, tratando-se valores de diferentes granularidades conforme explicado anteriormente.
- Como a classe fonte é bitemporal precisamos especificar, através da cláusula VALID, em que tempo válido o predicado será verificado — cada instante de tempo de transação corresponde a um histórico dos dados e, por conseguinte, para um mesmo tempo de transação o predicado pode ser verdadeiro em alguns instantes de tempo válido e falso em outros.
- A cláusula VALID AT indica que o predicado deverá ser testado apenas em um ponto de tempo — no exemplo %1992/05%.
- O resultado desta consulta é o conjunto de instantes de tempo de transação correspondentes aos estados do BD onde existe uma instância de *EMPREGADO(BT,escada,ano)* cuja componente *NOME* teve o valor "Marco" e a componente *CIDADE* teve o valor "Curitiba" em maio de 92.

4.3.1.3. Quando a revista "Veja" foi incluída no BD?

FIRST_INSTANT (**TWHEN** *r.Título = "Veja"*
VALID SOMETIME DURING ALLTIME
FROM *r in Revista)*

Comentários:

- Como existe apenas uma classe *REVISTA*, não é necessário especificar as suas características temporais.
- Nesta consulta incluímos uma expressão temporal predefinida da linguagem, ALLTIME. Outras expressões temporais predefinidas (NOW e TTIME) serão introduzidas em exemplos posteriores. ALLTIME indica que o predicado será testado em todos os instantes de tempo válido de cada estado do BD.
- A cláusula SOMETIME indica que serão selecionados os instantes de tempo de transação referentes aos estados do BD em que o predicado seja satisfeito em pelo menos um momento do tempo válido especificado ao invés de durante todo o tempo válido especificado. Esta cláusula corresponde a um quantificador existencial nos eixos de tempo.
- A consulta TWHEN retorna um elemento temporal contendo todos os instantes de tempo de transação, relativos a estados do BD onde existiu uma revista que teve o título "Veja" ao menos por um instante de tempo válido.
- FIRST_INSTANT retorna o primeiro instante do elemento temporal resultante da consulta TWHEN.

4.3.1.4. Em que estados do BD consta que a revista "Manchete" custou mais que 100 antes de Lauro tornar-se seu editor?

TWHEN *r.Título = "Manchete" and r.Preço.Valor > 100*
VALID SOMETIME BEFORE *r.Editor = "Lauro"*
FROM *r in Revista*

Comentários:

- Nesta consulta o tempo válido onde será testado se a revista Manchete custava mais que 100 (primeiro predicado) será determinado por um segundo predicado (Lauro ser seu editor) note que em estados distintos o tempo válido poderá ser diferente.
- Em cada estado do BD, o segundo predicado é testado e determina um elemento temporal contendo os instantes de tempo válido em que é verdadeiro (caso não seja verdadeiro em nenhum instante, o estado é descartado). O instante de tempo de transação correspondente a um estado qualquer do BD só será selecionado se o primeiro predicado for verificado em pelo menos um instante (SOMETIME) anterior ao primeiro instante (BEFORE) do elemento temporal determinado pelo primeiro predicado.
- Consideremos a figura 4.1. Esta figura representa a situação de um BDT em relação ao primeiro e segundo predicados. Os eixos horizontal e vertical representam tempo válido

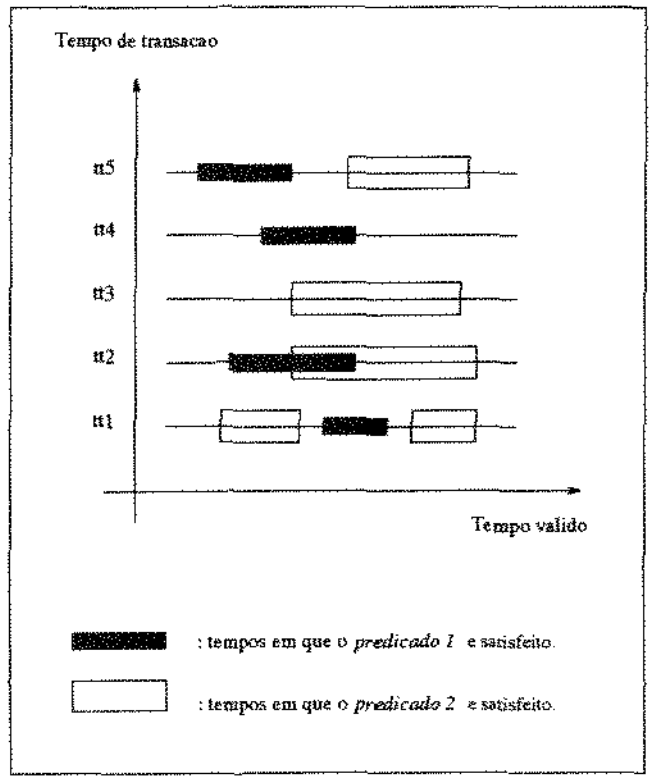
Tempo de transacao
tt5
tt4
tt3
tt2
tt1
Tempo valido
: tempos em que o predicado 1 e satisfeito.
: tempos em que o predicado 2 e satisfeito.

Figura 4.1:

e tempo de transação, respectivamente. O BD contém cinco estados correspondentes aos tempos de transação tt_1 a tt_5. Para este BD, a resposta à consulta seria um elemento temporal contendo apenas tt_2 e tt_5.

□

4.3.2 VWHEN

VWHEN realiza a operação correspondente a TWHEN, considerando-se tempo válido. A sintaxe da consulta VWHEN é:

```
VWHEN   predicado1
                                  AT
[INDB   [SOMETIME]  {  DURING  }  {  expressão_temporal1  }  ]
                                  {  predicado2           }
                       BEFORE
                       AFTER
[FROM   conjuntos/listas/classes fonte                       ]
```

Esta consulta retorna os instantes de tempo válido em que o *predicado*$_1$ é satisfeito nos estados do BD especificados na cláusula INDB. Caso o *predicado*$_1$ não envolva dados com suporte a tempo de transação a cláusula INDB não deve ser especificada. Assim como na consulta TWHEN, os instantes de tempo consecutivos são agrupados em intervalos, e a resposta é expressa na granularidade mais grossa possível.

Exemplos:

4.3.2.1. Quando Cláudio morou em Campinas?

VWHEN *e.Nome = "Cláudio" and e.Endereço.Cidade = "Campinas"*
FROM *e in Empregado*

Comentários:

- Utilizamos a classe primitiva da família — *EMPREGADO (TV,escada,ano)* — pois não foram especificadas as características temporais da classe fonte.
- Como a classe pesquisada é de tempo válido, não se especifica a cláusula INDB.
- Esta consulta retorna os instantes de tempo válido em que consta que algum objeto da classe fonte teve nome igual a Cláudio e cidade do endereço igual a Campinas.

4.3.2.2. Quando Ricardo morou em Belo Horizonte, de acordo com o conhecimento atual?

VWHEN *e.Nome = "Ricardo" and e.Endereço.Cidade = "Belo Horizonte"*
INDB AT NOW
FROM *e in Empregado(BT,escada.ano)*

Comentários:

- Como a classe fonte é bitemporal, é necessário especificar em que tempos de transação o predicado será testado — o predicado pode ser verdadeiro em um tv_1 de um estado e não o ser no mesmo tv_1 de outro estado.
- NOW é uma expressão temporal predefinida da TOOL que representa o tempo corrente. Portanto, o predicado será verificado no estado mais recente do BD.
- A consulta retorna os instantes de tempo válido em que, no último estado do BD, consta que algum objeto da classe fonte teve nome igual a Ricardo e cidade do endereço igual a Campinas.

4.3.2.3. Quando, segundo os dados da década de 80, Paulo foi editor de Veja?

VWHEN *r.Título = "Veja" and r.Editor = "Paulo"*
INDB DURING INTERVAL *(%1980%, %1989%)*
FROM *r in Revista*

Comentários:

- Embora a granularidade de *REVISTA* seja mensal, esta consulta pode ser executada sem problemas.
- Esta consulta recupera os instantes de tempo válido em que Paulo consta com editor de Veja em todos os estados do BD de 1980 a 1989.

4.3.2.4.Quando a Manchete teve preço superior a 100, nos estados em que Lauro era seu editor?

VWHEN *r.Título = "Manchete" and r.Preço.Valor > 100*
INDB DURING *r.Editor = "Lauro"*
FROM *r in Revista*

Comentários:

- Esta consulta é análoga ao exemplo 4.3.1.4, da consulta TWHEN, invertendo-se os papéis dos eixos de tempo de transação e tempo válido.
- O raciocínio para resolver esta consulta é similar ao utilizado naquele exemplo. Inicialmente, para cada tempo válido tv_i obtém-se em que estados do BD o segundo predicado — Lauro editor de Manchete — é verdadeiro (se não for verdadeiro em nenhum estado descarta-se este tv_i), resultando para cada tv_i um conjunto de tempos de transação TT_i. Um tv_j fará parte da resposta da consulta se, em todos os tempos de transação de TT_j, o primeiro predicado — Manchete com preço menor que 100 — também for verificado para o mesmo tempo válido tv_j.

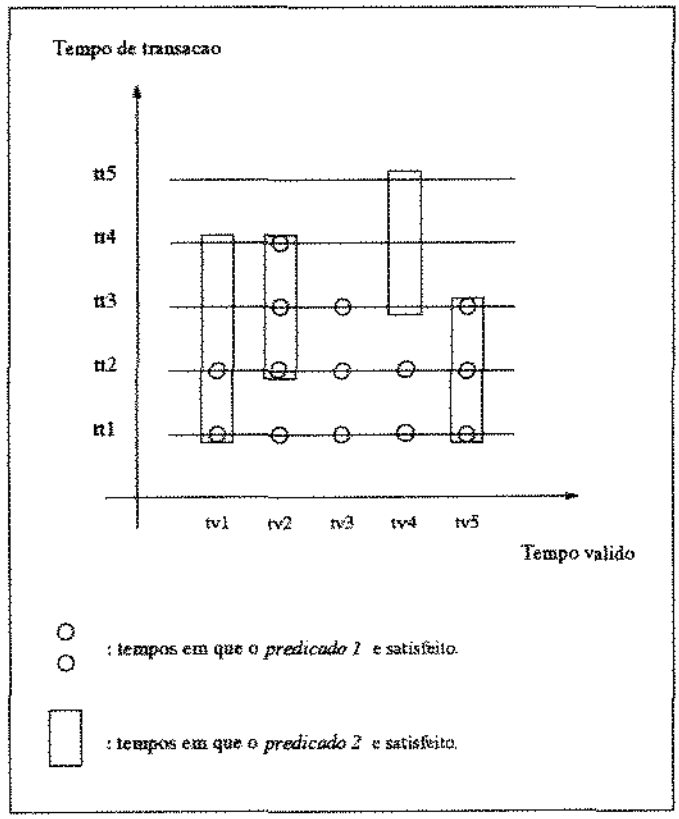
Tempo de transacao
tt5
tt4
tt3
tt2
tt1
tv1
tv2
tv3
tv4
tv5
Tempo valido
: tempos em que o *predicado 1* e satisfeito.
: tempos em que o *predicado 2* e satisfeito.

Figura 4.2:

- Consideremos a figura 1.2. Esta figura representa a situação de um BDT com cinco estados em relação ao primeiro e segundo predicados. Os eixos horizontal e vertical representam tempo válido e tempo de transação, respectivamente. São considerados na figura cinco instantes de tempo de válido (te_1 a te_5). Para este BD a consulta resultaria em um elemento temporal contendo apenas os tempos válido te_2 e te_5.

□

Embora a consulta VWHEN com a cláusula INDB determinada por um predicado possa parecer confusa ao usuário e, conseqüentemente deva ser utilizada menos freqüentemente, decidimos manter esta possibilidade por homogeneidade no tratamento das duas operações WHEN.

4.3.3 Extensão da Cláusula FROM

Observe a diferença semântica entre as consultas a seguir:

a. Segundo os dados atuais, quando houve uma revista com preço menor que 1000 ao mesmo tempo em que seu editor pesava mais que 100?

b. Segundo os dados atuais, quando houve uma revista com preço menor que 1000 ao mesmo tempo em que seu editor em junho de 1988 pesava mais que 100?

A primeira consulta pode ser expressa facilmente, seguindo o raciocínio utilizado nos exemplos das seções anteriores, da seguinte forma:

VWHEN *r.Preço.Valor < 1000 and r.Editor.Peso > 100*
INDB AT NOW
FROM *r in Revista*

A segunda consulta, porém, não pode ser facilmente expressa. A razão disto é que não temos meios de indicar quando houve a ligação de composição *REVISTA EDITOR*. Quando não havia suporte a tempo, a cláusula FROM especificava inequivocamente as relações de composição para cada consulta. Com a introdução do tempo, isto não mais acontece. Em cada tempo (válido e de transação) as ligações composto componente podem estar diferentes, o que torna esta cláusula inexata. A solução adotada para este problema será a extensão da cláusula FROM de maneira a permitir a especificação do tempo (válido e de transação) de cada ligação composto componente. Para tanto, utilizamos as cláusulas INDB e VALID, já introduzidas anteriormente. A segunda consulta poderia então ser expressa da seguinte forma:

VWHEN *r.Preço.Valor < 1000 and c.Peso > 100*
INDB **AT NOW**
FROM *r in Revista*
c in r.Editor **INDB AT NOW**
VALID AT *%1988/06%*

Esta consulta tem a seguinte interpretação. Recupere os instantes de tempo válido em que, no último estado do BD:

i. algum objeto R_1 da classe *REVISTA* teve o objeto componente *PREÇO* com valor menor que 1000 e

ii. seu editor no tempo (*NOW, %1988/06%*), teve peso maior que 100.

Portanto, a cláusula FROM servirá também para indicar em que tempos de transação é válido houve certas ligações entres objetos componentes e compostos.

Novamente, a cláusula INDB (VALID) deve ser especificada apenas para dados com suporte a tempo de transação (tempo válido).

Vejamos alguns exemplos de consultas WHEN com a cláusula FROM estendida para esclarecermos seu funcionamento.

Exemplos:

4.3.3.1. Segundo dados de agosto de 89, em que épocas o editor da Exame na década de 70 (segundo dados de 1980) teve peso igual a 70?

```
VWHEN   r.Título = "Exame" and e.Peso = 70
INDB    AT %1989/08%
FROM    r in Revista
        e in r.Editor      INDB AT %1980%
                           VALID DURING INTERVAL (%1970%,%1979%)
```

Comentários:

- A consulta retorna os instantes de tempo válido em que, em todos os estados do BD em agosto de 89, houve pelo menos uma revista com título "Exame" e cujo editor em (%1980%,%1970%...%1979%) pesava 70.
- Note que o editor em questão (E_1) deve ter constado em todos os estados do BD em 1980 como tendo sido editor de Exame durante toda a década de 70. Caso conste em algum estado do BD em 1980 que E_1 não foi editor de Exame em algum instante da década de 70, a consulta resultará em um elemento temporal vazio.

4.3.3.2. Em que estados do BD consta que algum dos editores da Placar no período de 1975 a 1985 morou em Curitiba no ano de 1990?

```
TWHEN   r.Título = "Placar" and e.Endereço.Cidade = "Curitiba"
VALID   AT %1990%
FROM    r in Revista
        e in r.Editor   INDB AT NOW
                        VALID SOMETIME DURING
                          INTERVAL (%1975%,%1985%)
```

Comentários:

- A especificação de SOMETIME na cláusula FROM indica que serão considerados os objetos que em ao menos um momento no intervalo 1975...1985 foram editores de revista.

- A consulta recupera os instantes de tempo de transação correspondentes aos estados do BD em que consta que no ano de 1990 houve ao menos uma revista com o título "Placar" e pelo menos um de seus editores no período 1975...1985 morava em Curitiba.

□

A cláusula FROM, além de indicar as relações de composição relevantes para a consulta, especifica ainda as classes, conjuntos ou listas referentes aos níveis mais altos destas composições (nos exemplos acima isto é efetuado na linha **FROM** *r in Revista*). Esta linha indica qual a classe fonte de onde serão retirados os objetos que serão a origem das composições consideradas na consulta. Portanto, a única indicação temporal possível neste caso da cláusula FROM é a determinação de quando os objetos de origem estiveram no BD usando para isto a cláusula INDB. Qualquer referência a tempo válido neste caso não teria sentido. Vejamos um exemplo deste caso.

Exemplo:

4.3.3.3. Segundo dados atuais, quando alguma das revistas cadastradas em 1975 teve preço menor que 100?

VWHEN *r.Preço.Valor* < *100*
INDB AT NOW
FROM *r in Revista* **INDB AT** *%1975%*

□

Observando os exemplos anteriores, notamos que a interpretação deste caso da cláusula FROM subentende a seguinte especificação:

FROM ...INDB SOMETIME DURING ALLTIME

Ou seja, são considerados todos os objetos que constaram em pelo menos um estado do BD. Como veremos na seção 4.7 este será o *default* para este caso da cláusula FROM.

4.4 Consultas que Retornam Dados em Função do Tempo

A seção anterior mostra consultas que retornam valores de tempo. Uma consulta pode igualmente recuperar uma parte do BD (e.g., podemos recuperar dados de certas classes, e destas classes apenas alguns objetos). Com a associação dos dados ao tempo em que foram válidos e/ou que estiveram no BD, dois novos tipos de restrições tornam-se necessários restrição no tempo válido e restrição no tempo de transação. Estas restrições podem ser utilizadas para obter estados passados do BD e para recuperar dados válidos em certos instantes.

Como dissemos no capítulo 2, a operação de restrição dos dados a serem obtidos baseada nos tempos associados a estes dados pode ser denominada fatiamento temporal. Em nosso modelo homogeneizamos o tratamento de tempo válido e de transação, definindo operações similares para

fatiamento temporal em ambos os eixos de tempo, ao contrário do que ocorre com os modelos analisados no capítulo 2. Com isto, o operador de manipulação relativo ao tempo de transação torna-se mais flexível que em outros modelos, como Jensen [JMR91] e Snodgrass [Sno87]. Nestes modelos é possível apenas se recuperar um único estado do BD. Em nosso modelo esta operação é generalizada, sendo possível a recuperação de um ou mais estados passados.

As operações de fatiamento temporal nos eixos de tempo de transação (TSLICE) e de tempo válido (VSLICE) foram implementadas na linguagem TOOL estendendo-se as consultas padrões ao BD com duas novas cláusulas:

```
Consulta-padrão  TSLICE  { AT | DURING | AFTER | BEFORE }  expressão_temporal1
                 VSLICE  { AT | DURING | AFTER | BEFORE }  expressão_temporal2
```

A cláusula VSLICE deve ser especificada apenas quando a *consulta-padrão* retornar dados com suporte a tempo válido. Analogamente, a cláusula TSLICE só deve ser especificada quando a *consulta-padrão* retornar dados com suporte a tempo de transação. Caso estas restrições não sejam respeitadas a consulta resultará em erro.

As cláusulas TSLICE e VSLICE agem de forma totalmente independente. A cláusula TSLICE é executada em primeiro lugar e restringe os dados resultantes da consulta padrão àqueles pertencentes aos estados do BD definidos na expressão_temporal$_1$. A cláusula VSLICE restringe os dados resultantes àqueles válidos na expressão_temporal$_2$.

Assim como fizemos com as operações TWHEN e VWHEN utilizamos exemplos para esclarecer o funcionamento de TSLICE e VSLICE.

Exemplos:

4.4.1. Quais os objetos da classe DINHEIRO de tempo de transação existentes no BD em 27 de maio de 1979?

Dinheiro **TSLICE AT** *%1979/05/27%*

Comentários:

- Como a classe *DINHEIRO* primitiva é de tempo de transação, não se especifica a cláusula VSLICE.
- Esta consulta retorna os dados da classe *DINHEIRO(TT)* referentes aos estados do BD em 27/05/79.

4.4.2. Forneça um histórico da classe REVISTA de 1980 até hoje.

Revista **TSLICE AT NOW**
VSLICE DURING INTERVAL (*%1980%***, NOW)**

Comentários:

- Como dissemos anteriormente, NOW é uma expressão temporal predefinida que representa o tempo corrente.
- A cláusula TSLICE seleciona o último estado do BD, i.e., os dados históricos das revistas corresponderão à visão mais atualizada dos dados.
- A cláusula VSLICE restringe os dados das revistas aos válidos entre 1980 e hoje. Portanto, são desprezados os dados anteriores a 1980 e os válidos no futuro (previsões).
- A consulta retornará os dados das instâncias que constem no estado mais atualizado da classe *REVISTA*, válidos entre 1980 e hoje.

4.4.3. Forneça os dados das revistas, segundo se conhecia antes de 18/06/73, com os valores válidos na época.

Revista **TSLICE BEFORE** *%1973/06/18%*
VSLICE AT TTIME

Comentários:

- TTIME é uma expressão temporal predefinida que indica o tempo de transação de cada estado. Assim, para o estado referente ao tempo de transação tt_i, VSLICE irá recuperar apenas os dados válidos em tt_i.
- Recupera todos os estados da classe *REVISTA* anteriores a 18/06/73 (TSLICE) sendo que em cada estado serão recuperados apenas os valores considerados correntes em cada estado, i.e., válidos no instante considerado o presente em cada estado (VSLICE).

4.4.4. Liste os nomes que constam no BD em 27/05/79 de todos que foram editores de revista.

Revista.Editor.Nome **TSLICE AT** *%1979/05/27%*
VSLICE DURING ALLTIME

Comentários:

- *Revista.Editor* recupera todos os objetos que foram editores de revista em algum tempo; *Revista.Editor.Nome* recupera todos os nomes destes editores.
- A cláusula TSLICE restringe os objetos resultantes aos existentes em algum estado do BD em 27/05/79, e os dados destes objetos aos relativos a estes estados.
- A cláusula VSLICE recupera todos os dados de cada objeto em cada estado de 27/05/79.

4.4.5. Forneça um histórico das revistas entre 1990 e 1992, para os estados do BD em que a revista "Saúde" esteve no BD.

Revista **TSLICE DURING (** **TWHEN** *r.Título = "Saúde"*
VALID SOMETIME DURING ALLTIME
FROM *r in Revista*)
VSLICE DURING INTERVAL (*%1990%,%1992%*)

Comentários:

- Como uma consulta TWHEN é uma expressão temporal, ela pode ser utilizada como argumento de TSLICE. Da mesma forma, uma consulta VWHEN pode servir de argumento para uma consulta VSLICE. A consulta TWHEN retorna os instantes de tempo de transação em que a revista Saúde constava no BD.
- A cláusula TSLICE restringe os dados da classe *REVISTA* aos relativos aos instantes de tempo de transação recuperados em TWHEN.
- Por fim, a cláusula VSLICE restringe os dados obtidos aos válidos entre 1990 e 1992.

4.4.6. Quais os dados dos empregados enquanto Letícia era a editora de Nova?

Empregado(BT,escada,ano) **TSLICE AT NOW**
VSLICE (**VWHEN DURING** *r.Editor = "Letícia" and r.Título = "Nova"*
INDB AT NOW
FROM *r in Revista*)

Comentários:

- A cláusula TSLICE seleciona o estado corrente do BD, portanto teremos os dados mais atualizados da classe fonte.
- A consulta VWHEN recupera o tempo válido em que Letícia foi editora da revista Nova, segundo os dados do estado atual do BD.
- Finalmente, a cláusula VSLICE restringe os dados da classe fonte aos válidos no tempo resultante da consulta VWHEN.

4.4.7. Quais os dados da classe REVISTA nos estados do BD em que todas revistas tinham preço < 1000 enquanto Juliano editava a revista Exame?

Revista **TSLICE DURING (** **TWHEN** *for all r in Revista : r.Preço < 1000*
VALID DURING *exists r in Revista : r.Título = "Exame" and r.Editor = "Juliano"*)
VSLICE DURING **ALLTIME**

Comentários:

- A consulta TWHEN é similar ao exemplo 4.3.1.4, e é resolvida da mesma forma. Esta consulta retorna os instantes de tempo de transação que satisfazem as exigências determinadas na consulta. Se a consulta fosse aplicada sobre o BD representado na figura 4.3 a resposta obtida seria o ponto de tempo tt_5.

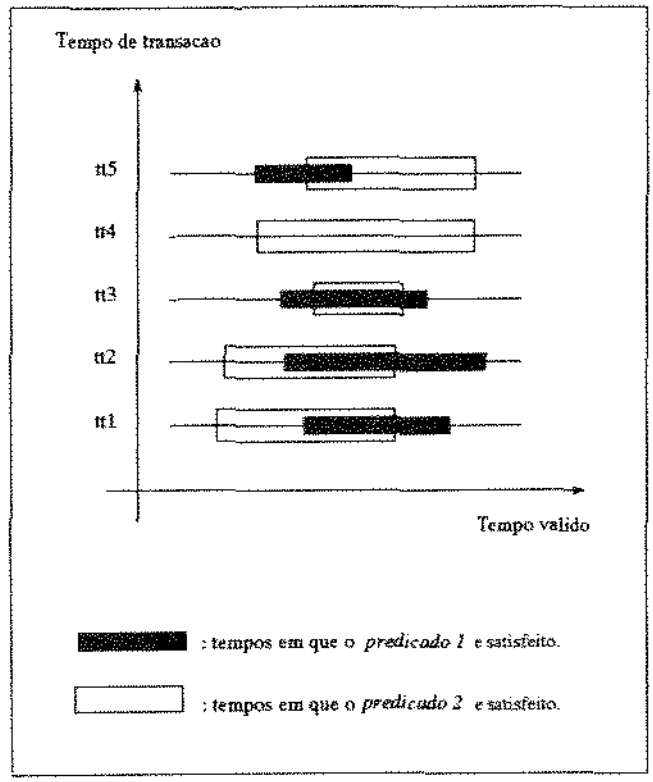
Tempo de transacao
tt5
tt4
tt3
tt2
tt1
Tempo valido
: tempos em que o predicado 1 e satisfeito.
: tempos em que o predicado 2 e satisfeito.

Figura 4.3:

- A cláusula TSLICE recupera os estados do BD relativos aos instantes de tempo de transação resultantes da consulta TWHEN.
- A cláusula VSLICE especifica que serão recuperados todos os dados da classe em cada estado.

□

4.5 Seleção Temporal

Até aqui não utilizamos a consulta base mais poderosa da linguagem O2Query, o bloco SELECT-FROM-WHERE (SFW). Assim como as demais consultas da O2Query, o bloco SFW também é estendido na TOOL com as cláusulas TSLICE e VSLICE para a indicação do tempo a que se referem os dados a serem recuperados. Além disto, de forma análoga às consultas TWHEN e VWHEN, a cláusula FROM deve ser estendida do modo especificado na seção 4.3.3. Por fim, é necessário que seja especificado em que tempo o predicado de seleção será testado. Para isto serão utilizadas as cláusulas INDB e VALID, já descritas. A sintaxe desta consulta, que denominamos **seleção temporal**, é então a seguinte:

```
SELECT alvo                    cláusula-TSLICE
                               cláusula-VSLICE
[cláusula-FROM estendida]
[WHERE predicado]              [cláusula-INDB]
                               [cláusula-VALID]]
```

Inicialmente são delimitados as listas/conjuntos/classes alvo. Isto é efetuado a partir da cláusula FROM. A seguir, os objetos do "alvo" são selecionados de acordo com a cláusula WHERE. Finalmente, é realizado o fatiamento do "alvo".

Passamos agora a alguns exemplos de seleção temporal para uma melhor compreensão de seu funcionamento.

Exemplos:

4.5.1. Quais os editores entre 1985 e 1990, da revista que em 1980 se chamava Senhor?

```
SELECT  e   TSLICE DURING ALLTIME
            VSLICE DURING ALLTIME
FROM    r in Revista
        e in r.Editor   INDB AT NOW
                        VALID SOMETIME DURING
                        INTERVAL (%1985%,%1990%)
WHERE   r.Título = "Senhor"  INDB AT NOW
                             VALID AT %1980%
```

Comentários:

- A cláusula FROM seleciona os objetos da classe *REVISTA* e os objetos que foram editores de revista em pelo menos um momento entre 1985 e 1990, segundo o estado mais recente do BD.

- A cláusula WHERE restringe os objetos selecionados em FROM às revistas cujo título no ano de 1980 é "Senhor", segundo o estado atual do BD.

- SELECT recupera apenas os editores destas revistas.

- Finalmente, é executado o fatiamento temporal.

Observe como o significado desta consulta seria alterado caso a especificássemos na forma:

```
SELECT r.Editor                TSLICE AT NOW
                               VSLICE DURING INTERVAL (%1985%,%1990%)
FROM r in Revista
WHERE r.Título = "Senhor"      INDB AT NOW
                               VALID AT %1980%
```

Esta consulta recupera os dados dos editores apenas no estado mais recente do BD e somente no intervalo 1985...1990. No entanto, as cláusulas de fatiamento agem sobre o resultado "final" da consulta e não servem para especificar o tempo em que ocorreu a ligação de composição entre os objetos. Portanto, os editores que farão parte da resposta serão selecionados de forma semelhante ao exemplo 4.4.4 — serão recuperados todos os objetos que foram, em pelo menos um instante, editores das revistas que satisfazem o predicado.

4.5.2. Segundo dados de 1980, quais revistas custavam mais de 500 quando editadas por Adauto?

```
SELECT r   TSLICE DURING ALLTIME
           VSLICE DURING ALLTIME
FROM r in Revista
WHERE      r.Preço.Valor > 500   INDB AT %1980%
                                 VALID DURING r.Editor = "Adauto"
```

Comentários:

- Serão selecionadas as revistas que em todos os estados do BD durante o ano de 1980 tinham preço superior a 500 (predicado 1) em todos os instantes em que foram editadas por Adauto (predicado 2).

- Na figura 4.4 vemos um exemplo da determinação dos tempos em que o predicado 1 deve ser testado para uma revista R_1 de um BDT qualquer.

4.5.3. Que revistas custavam mais que 500 em 1980, nos estados em que eram editadas por Adauto?

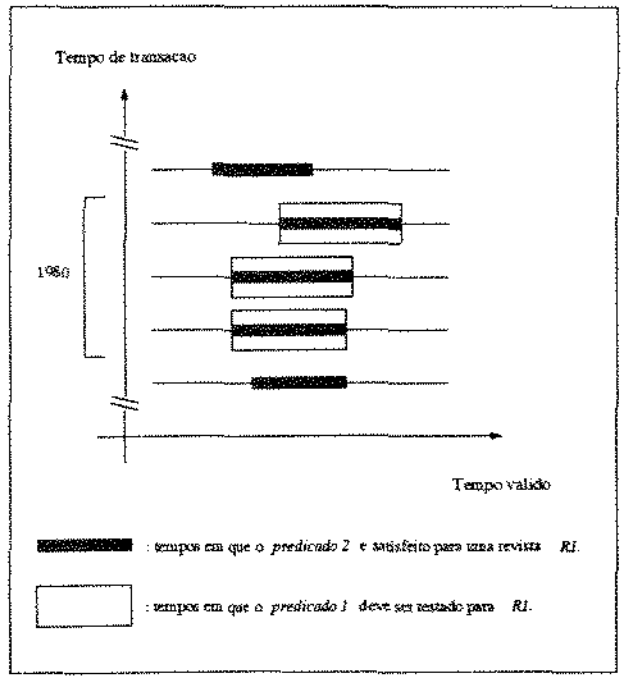

Figura 4.4:

SELECT *r* **TSLICE DURING ALLTIME**
VSLICE DURING ALLTIME
FROM *r in Revista*
WHERE *r.Preço.Valor > 500* **INDB DURING** *r.Editor = "Adauto"*
VALID AT *%1980%*

Comentários:

- Nesta consulta repetimos o exemplo 4.5.2, invertendo os papéis dos eixos de tempo de transação e tempo válido.

- Serão selecionadas as revistas que em cada instante de 1980 foram editadas por Adauto (predicado 1) em todos os estados do BD em que tinham preço superior a 500 (predicado 2).

- Na figura 4.5 vemos um exemplo da determinação dos tempos em que o predicado 1 deve ser testado para uma revista R_1 de um BDT qualquer.

4.5.4 Selecione os dados atuais de todos que foram editores de revistas em (tt_1, tv_1) *e moraram em* (tt_2, tv_2) *na cidade que se chamava Vila Rica em 1789.*

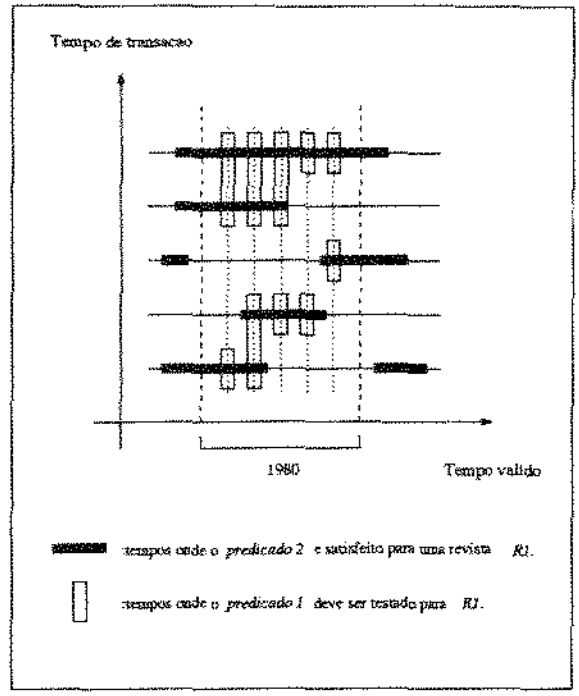

Figura 4.5:

SELECT *c* **TSLICE AT NOW**
VSLICE DURING ALLTIME
FROM *r in Revista*
e in r.Editor **INDB AT** tt_1
VALID AT tv_1
e in e.Endereço.Cidade **INDB AT** tt_2
VALID AT tv_2
WHERE *c* = "*Vila Rica*" **INDB AT NOW**
VALID AT 1789

Comentários:

- O termo *c in c.Endereço.Cidade* **INDB AT** tt_2 **VALID AT** tv_2 indica que as ligações de composição entre *CIDADE* e *ENDEREÇO* e entre *ENDEREÇO* e *EDITOR* devem existir no mesmo tempo válido e de transação, no caso tt_2 e tv_2.

- É interessante observar que as ligações de composição existentes na cláusula FROM podem corresponder a tempos completamente distintos — na consulta, o tempo de ligação *REVISTA EDITOR* é diferente de *EDITOR ENDEREÇO CIDADE*.

- A consulta recupera todos os dados do estado atual do BD dos objetos componentes (como editores) de instâncias da classe *REVISTA* em tt_1,tv_1, cujo componente *CIDADE* em tt_2,tv_2 tinha o nome "Vila Rica" em 1789.

□

4.6 Outras Consultas

Nesta seção, introduzimos as alterações no funcionamento de consultas sobre agregados e resultados de outras consultas.

As consultas base que manipulam valores atômicos (numéricos e booleanos) não sofrem nenhuma alteração, continuando a atuar somente com dados sem suporte a tempo. Logo, os argumentos destas consultas base devem ser restritos a dados atemporais.

As consultas que têm como argumentos listas ou conjuntos são estendidas para poder operar com dados com suporte a tempo. Estas consultas passam a funcionar como se executadas sobre os dados relativos a cada tempo de transação e tempo válido de seus argumentos. Quando a consulta envolver mais de um conjunto (lista), os argumentos devem ser da mesma categoria temporal. Como as demais consultas que lidam com dados temporais, estas consultas foram estendidas com as cláusulas VSLICE e TSLICE.

Exemplos:

4.6.1. Quais as médias de peso em 1970, segundo dados de 1980, entre todos que já foram editores de revista?

```
AVG ( SELECT  p  TSLICE DURING ALLTIME
                 VSLICE DURING ALLTIME
      FROM    r  in Revista
              p  in r.Editor.Peso  INDB AT NOW
                                   VALID SOMETIME DURING ALLTIME )
TSLICE AT 1980
VSLICE AT 1970
```

Comentários:

- A cláusula FROM seleciona todos os objetos que, segundo o estado atual do BD, foram *PESO* de objetos que em algum momento foram *EDITOR* de alguma revista. A cláusula SOMETIME tem efeito tanto sobre a ligação *EDITOR PESO* quanto sobre a ligação *REVISTA EDITOR*.
- A consulta interna recupera todos os dados em todos os estados do BD dos objetos determinados na cláusula FROM.
- A consulta AVG produz para cada tempo válido de 1970 de cada estado do ano de 1980 a média dos valores dos objetos *PESO* recuperados na consulta interna.
- O raciocínio utilizado nesta consulta é que a cada editor está associado um único objeto *PESO*, cujo valor pode variar com o tempo. Assim obteríamos para cada tempo válido e de transação a média dos pesos dos editores. Caso um editor esteja associado a mais de

um objeto *PESO*, os valores de cada um destes objetos seriam considerados no cálculo das médias, resultando em valores incorretos.

4.6.2. Recupere os dados de 1970 dos empregados que moraram em Fortaleza antes de 1985 e os dados de 1980 dos que moraram em Salvador antes de 1985.

```
DEFINE fort AS  SELECT e   TSLICE AT %1970%
                           VSLICE DURING ALLTIME
                FROM e in Empregado (BT,escada,ano)
                WHERE e.Endereço.Cidade = "Fortaleza"
                           INDB AT NOW
                           VALID SOMETIME BEFORE %1985%

DEFINE salv AS  SELECT e   TSLICE AT %1980%
                           VSLICE DURING ALLTIME
                FROM e in Empregado (BT,escada,ano)
                WHERE e.Endereço.Cidade = "Salvador"
                           INDB AT NOW
                           VALID SOMETIME BEFORE %1985%

fort UNION salv  TSLICE DURING ALLTIME
                 VSLICE DURING ALLTIME
```

Comentários:

- A consulta *fort* recupera todos os dados contidos no BD em 1970 dos objetos da classe *EMPREGADO(BT,escada,ano)*, cujo componente *ENDEREÇO* teve o componente *CIDADE* com o valor "Fortaleza" em ao menos um instante de tempo válido anterior a 1985, segundo os dados mais atualizados.

- A consulta *salv* recupera todos os dados registrados no BD em 1980 dos objetos da classe *EMPREGADO(BT,escada,ano)*, cujo componente *ENDEREÇO* teve o componente *CIDADE* com o valor "Salvador" em ao menos um instante de tempo válido anterior a 1985, segundo os dados mais atualizados.

- A consulta *UNION* efetua a união dos resultados da consultas *fort* e *salv*. Em cada estado do BD haverá a união dos objetos existentes em cada consulta.

□

Apesar de serem mantidos **métodos** como parte do esquema de estados passados do BD, não definimos uma política para lidar com estes métodos. Tais métodos têm seu escopo temporal de atuação limitado, visto que devem ser aplicados apenas a dados de acordo com o esquema a que pertencem, sem poder alterar estados passados do BD. O uso de métodos relativos ao esquema mais atualizado não sofre estas restrições.

O significado da operação de **extração de campos** em dados temporais já foi visto nos exemplos 4.4.4 e 4.5.2. Ao se extrair o componente de um objeto do tipo tupla, que tenha suporte a tempo, serão recuperados todos os objetos que foram componentes desta tupla em algum instante de tempo válido e de transação.

Em algumas consultas é necessário transformar dados de uma categoria temporal para outra categoria. Introduzimos com este propósito três novas consultas base na TOOL, a saber:

NOTIME (arg_1)
TRANSTIME (arg_2)
VALIDTIME (arg_3)

NOTIME converte dados com suporte a tempo de transação e/ou tempo válido em dados atemporais. Os dados resultantes serão aqueles referentes ao primeiro estado do BD (caso arg_1 tenha suporte a tempo de transação) e ao tempo válido mais antigo (caso arg_1 tenha suporte a tempo válido) que contenha dados de arg_1.

TRANSTIME converte dados bitemporais em dados com suporte apenas a tempo de transação. Em cada estado do BD serão mantidos apenas os dados referentes ao tempo válido mais antigo que contenha dados de arg_2.

VALIDTIME converte dados bitemporais em dados com suporte apenas a tempo válido. Permanecerão apenas os dados relativos ao primeiro estado do BD com dados de arg_3.

Exemplos:

4.6.3. Quais os nomes de todas as cidades que são parte de algum endereço atualmente?

```
DEFINE cid1 AS   SELECT c   TSLICE AT NOW
                            VSLICE AT NOW
                 FROM       c in Endereço (BT,escada,ano)
                            c in c.Cidade   INDB AT NOW
                                            VALID AT NOW

DEFINE cid2 AS   SELECT c   VSLICE AT NOW
                 FROM       c in Endereço (TV,escada,ano)
                            c in c.Cidade   VALID AT NOW

DEFINE cid3 AS   SELECT c
                 FROM       c in Endereço
                            c in c.Cidade

(cid3 UNION NOTIME (cid2)) UNION NOTIME (cid1)
```

Comentários:

- Nas consultas *cid1*, *cid2* e *cid3* recuperamos todos os objetos que são componentes atuais (como *CIDADE*) de objetos da família *ENDEREÇO*.

- Na consulta final realizamos a união dos três conjuntos obtidos, desconsiderando suas características temporais.

4.6.4. Quais as médias de peso, segundo o estado mais recente do BD, de todas as instâncias da família PESSOA?

```
DEFINE P1 AS  SELECT  peso  TSLICE AT NOW
                            VSLICE DURING ALLTIME
              FROM    p in  Pessoa (BT,escada,ano)
                      peso  in p.Peso  INDB AT NOW
                                       VALID AT NOW

DEFINE P2 AS  SELECT  peso VSLICE DURING ALLTIME
              FROM    p in Pessoa
                      peso in p.Peso VALID AT NOW

AVG ((VALIDTIME (P1) UNION P2) VSLICE DURING ALLTIME)
VSLICE DURING ALLTIME
```

Comentários:

- *P1* e *P2* obtêm os pesos do último estado do BD dos objetos da família *PESSOA*. Na cláusula FROM de *P2* não são especificadas as características temporais da classe fonte por se tratar da classe primitiva da família.
- A consulta UNION une os conjuntos *P1* e *P2*, decartando o suporte a tempo de transação de *P1*. Por fim, a consulta AVG obtém as médias dos pesos em cada tempo válido.

□

4.7 Defaults da Linguagem

Introduzimos uma série de *defaults* na linguagem TOOL visando dois objetivos básicos. O primeiro objetivo é facilitar a especificação de algumas consultas, desobrigando o usuário de especificar todas as cláusulas VALID, INDB, VSLICE e TSLICE possíveis nestas consultas. A segunda meta é permitir ao usuário consultar classes surgidas na temporalização considerando apenas as características temporais das classes primitivas de suas famílias, mantendo uma interpretação coerente para estas consultas.

O primeiro *default* da linguagem, visto neste capítulo, diz respeito às famílias de classes. Caso em uma consulta qualquer não sejam especificadas as características temporais de uma classe (o que a identificaria univocamente em sua família) será utilizada a classe primitiva da família desta classe. A utilização deste *default* foi vista em boa parte dos exemplos da seção anterior.

Pela característica funcional da linguagem, cada consulta pode conter diversas subconsultas como argumentos. Tal propriedade traz o inconveniente de precisarmos especificar as cláusulas

TSLICE e VSLICE diversas vezes numa consulta que seja composta de outras consultas. Em razão disto algumas consultas podem tornar-se longas e difíceis de serem manipuladas. Para evitar estes problemas convencionamos que em consultas onde o tempo referente a VSLICE e TSLICE das subconsultas for igual ao da consulta que as contém, é necessário especificar apenas o tempo desta última. Resumindo, as cláusulas TSLICE e VSLICE das consultas aninhadas são, por *default*, iguais às determinadas para a consulta externa.

Exemplos:

4.7.1. A consulta:

COUNT (*Revista* **TSLICE AT** *%1987%*
VSLICE AT TTIME)
TSLICE AT *%1987%*
VSLICE AT TTIME

Poderia ser especificada com a utilização dos *defaults* como:

COUNT (*Revista* **) TSLICE AT** *%1987%*
VSLICE AT TTIME

Esta consulta conta para cada estado do BD em 1987, o número de revistas cadastradas com existindo no tempo corrente da época.

4.7.2. Com a utilização de *defaults*, a consulta do exemplo 4.6.4 poderia ser expressa simplesmente como:

AVG (VALIDTIME (*P1***) UNION** *P2***) VSLICE DURING ALLTIME**

□

Quando a consulta mais externa não traz a especificação da cláusula TSLICE, o SGBD interpreta a consulta como sendo relativa ao estado atual do BD, i.e., *TSLICE AT NOW*. Esta interpretação segue o princípio de que, quando o usuário não especifica o tempo de uma consulta, ele espera obter os dados correntes. Além disto, caso a classe fonte seja uma classe derivada, i.e., originária da temporalização, cujo suporte a tempo de transação é decorrente deste processo, e o usuário deseja consultá-la considerando-a com as características temporais da classe primitiva, esta seria a melhor interpretação. Classes sem suporte a tempo de transação mantêm apenas o estado mais recente dos dados, que seria o estado recuperado ao utilizarmos este *default*.

No caso da consulta mais externa não trazer a especificação da cláusula VSLICE, a interpretação do SGBD é que a consulta se refere aos dados correntes em cada estado do BD, i.e., *VSLICE AT TTIME*. Esta interpretação segue os mesmos princípios da cláusula TSLICE. Quando o usuário não determina o tempo de uma consulta, supõe-se que ele deseja os dados correntes. Além disto, classes que não ofereçam suporte a tempo válido, mantêm apenas os dados considerados correntes.

Exemplos:

4.7.3. Quais os dados atuais da revista Placar?

SELECT *r*
FROM *r in Revista*
WHERE *r.Título = "Placar"* **INDB AT NOW**
VALID AT NOW

Os termos *TSLICE AT NOW* e *VSLICE AT TTIME* são acrescentados, por omissão, após a cláusula SELECT.

4.7.4. Com a utilização deste *default*, a consulta do exemplo 4.7.1 poderia ser expressa de forma ainda mais reduzida:

COUNT (*Revista* **) TSLICE AT** *%1987%*

Ao observarmos os exemplos das seções anteriores, podemos notar que este *default* poderia ser utilizado em um grande número destes exemplos, facilitando ao usuário a especificação de tais consultas.

□

Como as cláusulas VALID e INDB são utilizadas em diversas situações em consultas temporais, não definimos um *default* único para estas cláusulas. Apesar disto, procuramos não divergir dos princípios já descritos para o estabelecimento de *defaults* .

Quando as cláusulas INDB e VALID forem omitidas após a cláusula WHERE, o SGBD entenderá que o predicado de seleção deverá ser verificado nos tempos de transação e tempos válidos determinados para TSLICE e VSLICE, respectivamente. O raciocínio utilizado aqui é que as condições de seleção devem ser verificadas nos mesmos tempos de onde serão extraídos os dados. Além disto, esta interpretação não contraria os princípios estabelecidos para os *defaults* de fatiamento temporal. Assim, caso não seja indicada nenhuma cláusula temporal para uma consulta, as cláusulas TSLICE e VSLICE determinarão, por omissão, os dados correntes do último estado do BD e, por conseguinte, o predicado também será testado nos tempos relativos a estes dados.

Nos outros casos de utilização das cláusulas INDB e VALID, o SGBD interpreta a omissão destas como se o usuário tivesse especificado *INDB AT NOW* e *VALID AT TTIME*, exceto no caso especial da cláusula INDB visto na seção 4.3.3. Como dissemos naquela seção, a interpretação do SGBD neste caso é *INDB SOMETIME DURING ALLTIME*. Este *default* foi utilizado na maior parte dos exemplos das seções anteriores. Contudo, para mantermos os mesmos princípios na interpretação das cláusulas temporais omitidas, este *default* deve ser alterado quando a consulta não especificar nenhuma cláusula temporal. Neste caso, somente, o *default* para a cláusula INDB deverá ser idêntico ao caso comum desta cláusula, ou seja, *INDB AT NOW*.

Exemplos:

4.7.5. Quais as revistas cujos editores pesam 80?

a.
```
SELECT   r
FROM     r in Revista
         e in r.Editor
WHERE    e.Peso = 80
```

b.
```
SELECT   r  TSLICE AT NOW
            VSLICE AT TTIME
FROM     r in Revista   INDB AT NOW
         e in r.Editor  INDB AT NOW
                        VALID AT TTIME
WHERE    e.Peso = 80    INDB AT NOW
                        VALID AT TTIME
```

Na segunda consulta estão especificados os *defaults* utilizados pela omissão das cláusulas temporais, na primeira consulta.

□

4.8 Considerações Finais

Um de nossos objetivos ao definirmos uma linguagem de consulta para o modelo TOODM era o de estudar as novas operações possíveis em um modelo temporal OO. Procuramos transformar estas novas operações em uma linguagem de consulta real para obtermos uma visão mais concreta do poder de tais operações, bem como ter um maior contato com os problemas que um usuário final enfrentaria ao trabalhar com um SGBD baseado no TOODM.

Apesar de procurarmos apresentar todas as características da linguagem de maneira clara e mantendo coerência entre as diversas construções da linguagem, a apresentação foi informal. Não nos propusemos a apresentar uma definição formal das novas construções, optando por introduzí-las de forma intuitiva. Da mesma forma, não pretendemos provar a completeza desta linguagem.

Diversas extensões seriam desejáveis à linguagem TOOL. Estas extensões são mencionadas no capítulo seguinte.

Capítulo 5

Conclusão

Esta dissertação apresenta três contribuições principais, a saber:

- elaboração de uma ampla revisão bibliográfica sobre o tema de Bancos de Dados Temporais;
- produção de um modelo de dados temporal (TOODM), baseado no modelo OO; e
- especificação de uma linguagem de consulta (TOOL) para o modelo proposto.

A revisão bibliográfica é em si uma contribuição original. Ela considerou quase duas dezenas de modelos temporais, muitos dos quais apresentando consideráveis alterações no decorrer do tempo. A revisão considerou as principais propostas no campo de BDT's, publicadas até o ano de 1992. Cada um dos modelos foi descrito separadamente e todos os modelos foram analisados comparativamente com relação a variados aspectos ligados ao problema temporal.

O modelo TOODM aqui proposto apresenta as seguintes características:

- utiliza diversos conceitos encontrados nas propostas de BDT's vistas na revisão bibliográfica, em especial os conceitos de tempo válido e tempo de transação, as diversas categorias de dados temporais advindas destes conceitos e as várias formas de variação de valores de dados;
- considera questões, fundamentais em nosso entendimento, mas normalmente ignoradas na literatura, como a existência simultânea de vários tipos de dados temporais e a possibilidade de evolução de esquema; e
- considera as conseqüências da introdução dos conceitos temporais em questões da orientação a objetos, tais como herança, composição e identidade de objetos.

Em decorrência destas características, o modelo TOODM lida com alguns problemas inéditos na literatura, como a compatibilização de classes com diferentes propriedades temporais e a migração de objetos.

A linguagem de consulta TOOL para um SGBD baseado no TOODM possui as seguintes propriedades principais:

- contém construções para realizar as principais operações temporais de consulta existentes em outras propostas; e

- trata de modo homogêneo as diversas operações temporais — as cláusulas relativas a operações sobre o tempo de transação são análogas às cláusulas relativas ao tempo válido.

O TOODM e a TOOL apresentam as seguintes características, considerando os tópicos analisados no capítulo 2:

Modelo de dados base: Modelo OO.

Dimensões temporais: São suportadas duas dimensões temporais, uma relacionada a tempo válido e outra a tempo de transação.

Variação de valores: Os quatro tipos de variação de valores (escada, discreta, contínua e definida pelo usuário) são permitidas no modelo.

Evolução de esquema: Possibilita-se a evolução de esquema através da manutenção de esquemas independentes associados a cada estado do BD; o modelo trata da manutenção de esquemas de estados passados, das relações entre as classes afetadas com as alterações no esquema e do destino das instâncias destas classes.

Nível de registro de tempo e representação do tempo: Estas questões foram consideradas relativas à implementação ao invés ao modelo propriamente dito e, portanto, não foram discutidas em nosso modelo. A princípio consideramos que seria mais interessante associar valores de tempo ao nível de componentes de objetos e que as marcas de tempo poderiam ser pontos ou conjuntos de intervalos de tempo, a depender da variação de valores de cada classe.

Implementação: A dissertação não se preocupou com aspectos de implementação. Entretanto, algumas características do TOODM e da TOOL estão sendo atualmente implementados como uma camada do sistema O_2.

Tratamento de alterações: Pela manutenção das duas dimensões de tempo, são permitidas alterações ordinárias ou retroativas sem ocasionar perda de dados.

Operadores algébricos e linguagem de consulta: Propusemos a linguagem de consulta TOOL para o modelo, com as características acima descritas.

Podemos citar como seqüência natural ao trabalho desta dissertação as seguintes extensões:

- o tratamento da questão de comportamento na definição do TOODM;
- a definição formal da linguagem TOOL;
- a implementação do modelo TOODM e da linguagem TOOL; e
- algumas extensões necessárias à TOOL, a saber:
  - Consultas sobre o esquema: Como o modelo TOODM admite que o esquema de um BDT seja alterado com o passar do tempo, consultas sobre a definição do esquema em estados passados do BD podem tornar-se necessárias.

- Consultas sobre dados em esquemas variantes e sobre objetos migrantes: A recuperação dos dados de objetos que estiveram em classes com diferentes definições, seja pela evolução de esquema, seja pela migração do objeto, traria algumas complicações para a realização de consultas.
- Tratamento de métodos: A questão de passagem de mensagens quase não foi abordada na TOOL. Esta questão deve ser analisada tendo em vista principalmente o problema de evolução de esquema.
- Famílias de Classes: Classes de uma família são tratadas na TOOL como classes totalmente independentes. Entretanto, a ligação entre estas classes é importante e consultas que considerem esta ligação poderiam ser estudadas.
- Operações de Atualização: Parece-nos óbvio que as operações de atualização devem ser também estendidas para o modelo TOODM o significado destas operações é profundamente alterado ao considerarmos o paradigma temporal.

Bibliografia

[ABD+89] Malcon Atkinson, François Bancilhon, David DeWitt, Klaus Dittrich, David Maier, and Stanley Zdonik. The object-oriented database system manifesto. Technical Report 30-89, GIP Altair, Le Chesnay, France, September 1989.

[ABN87] T. Abbod, K. Brown, and H. Noble. Providing time-related constraints for conventional database systems. In *Proceedings of the International Conference on Very Large Data Bases*, pages 167–175, Brighton, 1987.

[AQ86] M. Adiba and N. Bui Quang. Historical multi-media databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 63–70, Kyoto, Japan, August 1986.

[Ari86] Gad Ariav. A temporally oriented data model. *ACM Transactions on Database Systems*, 11(4):499–527, December 1986.

[AS86] Ilsoo Ahn and Richard Snodgrass. Performance evaluation of a temporal database management system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 96–107, 1986.

[BCD89] François Bancilhon, Sophi Cluet, and Claude Delobel. A query language for the O2 object-oriented database system. Technical Report 35-89, GIP Altair, Le Chesnay, France, August 1989.

[BKKK87] Jay Banerjee, Won Kim, Hyoung-Joo Kim, and Henry F. Korth. Semantics and implementation of schema evolution in object-oriented databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 311–322, San Francisco, California, USA, May 1987.

[CC87] James Clifford and Albert Croker. The historical relational data model (HRDM) and algebra based on lifespans. In *Proceedings of International Conference on Data Engineering*, pages 528–537, 1987.

[CC88] James Clifford and Albert Croker. Objects in time. *IEEE Data Engineering*, 11(4):11–18, December 1988.

[CC89] Albert Croker and James Clifford. On completeness of historical relational data models. Technical report, New York University, New York, NY, USA, January 1989.

[Cha88] Surahit Chaudhuri. Temporal relationships in databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 160–170, Los Angeles, California, USA, 1988.

[CT85] James Clifford and Abdullah Uz Tansel. On an algebra for historical relational databases: Two views. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 247–265, Austin, Texas, USA, May 1985.

[CW83] James Clifford and David S. Warren. Formal semantics for time in databases. *ACM Transactions on Database Systems*, 8(2):214–254, June 1983.

[Dat88] C. J. Date. A proposal for adding date and time support to SQL. *SIGMOD Record*, 17(2):53–76, June 1988.

[EWK90] Ramez Elmasri, Gene T. J. Wuu, and Yeong-Joon Kim. The time index: An access structure for temporal data. In *Proceedings of the International Conference on Very Large Data Bases*, pages 1–12, Brisbane, Australia, 1990.

[Gad86] Shashi K. Gadia. Weak temporal relations. In *Proceedings of the 5th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 70–77, Cambridge, Massachusets, USA, March 1986.

[Gad88] Shashi K. Gadia. A homogeneous relational model and query languages for temporal databases. *ACM Transactions on Database Systems*, 13(4):418–448, December 1988.

[GIP89] GIP Altair, Le Chesnay, France. *The O2 Query Language User's Manual*, 1989.

[GM91] Dov Gabbay and Peter McBrien. Temporal logic and historical databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 423–430, Barcelona, Spain, September 1991.

[GS90] Himawan Gunadhi and Arie Segev. A framework for query optimization in temporal databases. In *Fifth International Conference on Statistical and Scientific Database Management*, pages 131–147, April 1990.

[GV85] Shashi K. Gadia and Jay H. Vaishnav. A query language for a homogeneous temporal database. In *Proceedings of the 4th ACM SIGACT-SGMOD Symposium on Principles of Database Systems*, pages 51–56, Portland, Oregon, USA, March 1985.

[GY88] Shashi K. Gadia and Chuen-Sing Yeung. A generalized model for a relational temporal database. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 251–259, Chicago, Illinois, USA, June 1988.

[JCG+92] Christian S. Jensen, James Clifford, S. K. Gadia, Arie Segev, and Richard T. Snodgrass. A glossary of temporal database concepts. *SIGMOD Record*, 21(3), September 1992.

[JMR91] Christian S. Jensen, Leo Mark, and Nick Roussopoulos. Incremental implementation model for relational databases with transaction time. *IEEE Transactions on Knowledge and Data Engineering*, 3(4):461–473, December 1991.

[JS92] Christian S. Jensen and Richard T. Snodgrass. Temporal specialization. In *Proceedings of International Conference on Data Engineering*, pages 594–602, 1992.

[KC86] Setrag N. Khoshafian and George P. Copeland. Object identity. In *Proceedings of ACM Conference on Object-Oriented Programming Systems, Language and Aplications*, Portland, Oregon, USA, September 1986.

[KC88] Won Kim and Hong-Tai Chou. Versions of schema for object-oriented databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 148–159, Los Angeles, California, USA, 1988.

[KL83] Manfred R. Klopprogge and Peter C. Lockemann. Modelling information preserving databases : Consequences of the concept of time. In *Proceedings of the International Conference on Very Large Data Bases*, pages 399–416, Florence, Italy, 1983.

[KRS90] W. Käfer, N. Ritter, and H. Schöning. Support for temporal data by complex objects. In *Proceedings of the International Conference on Very Large Data Bases*, pages 24–35, Brisbane, Australia, 1990.

[KS92] Wolfang Käfer and Harald Schöning. Realizing a temporal complex-object data model. In *SIGMOD*, pages 266–275, San Diego, California, USA, 1992.

[KYL90] Jinho Kim, Hodong Yoo, and Yoonjoon Lee. Design and implementation of a temporal query language with abstract time. *Information Systems*, 15(3):349–357, 1990.

[LDE+84] V. Lum, P. Dadam, R. Erbe, J. Guenauer, P. Pistor, G. Walch, H. Werner, and J. Woodfill. Designing DBMS support for the temporal dimension. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 115–130, Boston, MA, USA, June 1984.

[LJ88] Nikos A. Lorentzos and Roger G. Johnson. Extending relational algebra to manipulate temporal data. *Information Systems*, 13(3):289–296, 1988.

[LM90] T. Y. Cliff Leung and Richard R. Muntz. Query processing for temporal databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 200–208, Atlantic City, New Jersey, USA, May 1990.

[LRV88] Christophe Lécluse, Philippe Richard, and Fernando Velez. O2, an object-oriented data model. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 424–433, Chicago, Illinois, USA, June 1988.

[McK86] Edwin McKenzie. Bibliography : Temporal databases. *SIGMOD Record*, 15(4):40–52, December 1986.

[MNA87] N. G. Martin, Shamkant B. Navathe, and Rafi Ahmed. Dealing with temporal schema anomalies in history databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 177–184, Brighton, 1987.

[MP91] Roberto Maiocchi and Barbara Pernici. Temporal data management systems: A comparative view. *IEEE Transactions on Knowledge and Data Engineering*, 3(4):504–524, December 1991.

[MS87] Edwin McKenzie and Richard Snodgrass. Extending the relational algebra to support transaction time. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 467–478, San Francisco, California, USA, May 1987.

[MS90] Edwin McKenzie and Richard Snodgrass. Schema evolution and the relational algebra. *Information Systems*, 15(2):207–232, 1990.

[MS91] L. Edwin McKenzie and Richard T. Snodgrass. Evaluation of relational algebras incorporating time dimension on databases. *ACM Computing Surveys*, 23(4):501–543, December 1991.

[NA89] Shamkant B. Navathe and Rafi Ahmed. A temporal relational model and a query language. *Information Sciences*, 49(1):147–175, 1989.

[NR89] G. T. Nguyen and D. Rieu. Schema evolution in object-oriented database systems. *Data and Knowledge Engineering*, (4):46–67, 1989.

[PS87] Jason Penney and Jacob Stein. Class modification in GemStone Object-Oriented DBMS. In *Proceedings of ACM Conference on Object-Oriented Programing Systems, Language and Aplications*, pages 111–117, October 1987.

[SA85] Richard Snodgrass and Ilsoo Ahn. A taxonomy of time in databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 236–246, Austin, Texas, USA, May 1985.

[SA86] Richard Snodgrass and Ilsoo Ahn. Temporal databases. *IEEE Computer*, 19(9):35–42, September 1986.

[Sar90a] Nandlal L. Sarda. Algebra and query languages for a historical data model. *The Computer Journal*, 33(1):11–18, 1990.

[Sar90b] Nandlal L. Sarda. Extensions to SQL for historical databases. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):220–230, June 1990.

[SC91] Stanley Y. W. Su and Hsin-Hsing M. Chen. A temporal knowledge representation model OSAM*/T and its query language OQL/T. In *Proceedings of the International Conference on Very Large Data Bases*, pages 431–442, Barcelona, Spain, September 1991.

[SK86] Arie Shoshani and Kyoji Kawagoe. Temporal data management. In *Proceedings of the International Conference on Very Large Data Bases*, pages 79–88, Kyoto, Japan, August 1986.

[Sno92] Richard Snodgrass. TSQL: A design approach. White Paper, 1992.

[Sno86] Richard Snodgrass. Research concerning time in database, project summaries. *SIGMOD Record*, 15(4):19–39, December 86.

[Sno87] Richard Snodgrass. The temporal query language TQUEL. *ACM Transactions on Database Systems*, 12(2):247–298, June 87.

[Sno90] Richard Snodgrass. Temporal databases status and research directions. *SIGMOD Record*, 19(4):83–89, December 90.

[Soo91] Michael D. Soo. Bibliography on temporal databases. *SIGMOD Record*, 20(1):14–23, March 1991.

[SS87] Arie Segev and Arie Shoshani. Logical modelling of temporal data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 454–466, San Francisco, California, USA, May 1987.

[SS88] Arie Segev and Arie Shoshani. The representation of a temporal data model in the relational environment. *Lecture Notes in Computer Science*, 339:39–61, 1988.

[Tan86] Adullah Uz Tansel. Adding time dimension to relational model and extending relational algebra. *Information Systems*, 11(4):343–355, 1986.

[TAO90] Adullah Uz Tansel, M. Erql Arkun, and Gultikin Ozsoyoglu. Time-by-example query language for a historical data model. *The Computer Journal*, 33(1):11–18, 1990.

[TC90] Alexander Tuzhilin and James Clifford. A temporal relational algebra as a basis for temporal relational completeness. In *Proceedings of the International Conference on Very Large Data Bases*, pages 13–23, Brisbane, Australia, 1990.

[TG87] Adullah Uz Tansel and L. Garnett. Nested historical relations. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 284–293, San Francisco, California, USA, May 1987.

[WD92] Gene T. J. Wuu and Umeshwar Dayal. A uniform model for temporal object-oriented databases. In *Proceedings of International Conference on Data Engineering*, pages 584–593, 1992.