UNIVERSIDADE FEDERAL DA PARAÍBA - UFPB

CENTRO DE CIÊNCIAS E TECNOLOGIA - CCT

COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA - COPIN

DISSERTAÇÃO DE MESTRADO

REALIZAÇÃO DE CONSULTAS VISUAIS A BANCO DE DADOS TEMPORAIS

GILENE FERNANDES SANTOS

CAMPINA GRANDE, SETEMBRO DE 2000

Gilene Fernandes Santos

REALIZAÇÃO DE CONSULTAS VISUAIS A BANCO DE DADOS TEMPORAIS

Dissertação submetida ao Curso de Pós-Graduação em Informática do Centro de Ciências e Tecnologia da Universidade Federal da Paraíba, como requisito parcial para a obtenção do grau de Mestre em Informática.

Área de Concentração: **Ciência da Computação**

Linha: **Banco de Dados**

Ulrich Schiel

Orientador

Campina Grande, Setembro de 2000

S237R

Santos, Gilene Fernandes

Realização de Consultas Visuais a Banco de Dados Temporais

Dissertação de Mestrado, Universidade Federal da Paraíba, CCT, COPIN, Campina Grande, PB, Setembro de 2000

109 pág. Il. Orientador: Ulrich Schiel

1. Sistemas Visuais de Consultas
2. Banco de Dados Temporal

CDU – 681.3.07B

"...O verdadeiro tamanho de um homem
perante Deus é o tamanho de sua fé..."

Nelson Ned

Agradecimentos

Primeiramente a **Deus** que nos momentos mais difíceis foi a ele que recorri.

Aos **meus pais, irmãos e irmã** que sempre me apoiaram e incentivaram.

À **Dona Graça, Sr. Assis, Ana Cristina e Diana** pelo acolhimento que me deram e que até hoje estão sempre presentes na minha vida.

À todos os **meus amigos e amigas** de Campina Grande que sempre estiveram ao meu lado.

À todos os **meus colegas** da área que começaram comigo esta empreitada e que sempre me ajudaram nos momentos mais difíceis.

Ao meu **orientador Ulrich Schiel** e a **Sônia Fernandes** pelo apoio e conhecimento transmitido.

As funcionárias da COPIN, **Vera** e **Aninha**, pelo apoio e incentivo.

Aos professores da COPIN, pelos conhecimentos transmitidos e apoio na execução deste trabalho.

À **CAPES** pelo suporte financeiro durante todo o trabalho de Mestrado.

Resumo

O ambiente visual de consulta temporal, denominado TVQE (*Temporal Visual Environment Query – TVQE*), permite a formulação de consultas a banco de dados históricos. Este ambiente adota uma representação diagramática do esquema conceitual do banco de dados (incluindo classes e relacionamentos, temporais ou não) e uma "agenda gráfica" como metáfora de interação. O usuário pode formular tanto consultas convencionais como temporais, sendo que, para consultas temporais o ambiente oferece uma interação específica com operadores especiais utilizados para construção das condições temporais. Nesta dissertação, foram realizadas modificações e extensões no TVQE, além de um componente denominado Tradutor, que transforma as consultas formuladas pelo usuário no TVQE para SQL. As modificações realizadas estão relacionadas aos aspectos de interação do sistema com o usuário, tais como: a inclusão de um painel de mensagens e a sincronização de componentes gráficos (menus, ícones e painéis). As extensões realizadas foram: a formulação de consultas complexas como um novo tipo de consulta, a visualização do esquema gráfico do resultado da consulta e a migração do ambiente TVQE para um *applet* Java.

Abstract

The Temporal Visual Query Environment (TVQE) allows the formulation of queries to historical databases. This environment adopts a diagrammatic representation of the database schema (including classes and relationships, temporal or not) and a "graphic notebook" as interaction metaphor. The user can formulate conventional queries as well as temporal queries, for the temporal queries the environment offers a specific interaction with special operators used for the formulation of temporal conditions. In this dissertation, we present some modifications and extensions of TVQE, besides a component denominated Translator, that translates queries formulated by the user in the TVQE to SQL. The modifications are related to system interaction aspects with the user, such as: the inclusion of a messages panel and the syncronizing of graphic components (menus, icons and panels). The extensions were: the formulation of complex queries as a new query type, the visualization of the graphic schema of the query result and the migration of the environment TVQE to an Java applet.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1 - Introdução

Os sistemas de consultas a banco de dados, na sua maioria, requerem do usuário um conhecimento prévio de uma linguagem de acesso ao banco de dados (BD), por exemplo, SQL. Para o usuário casual isso pode se tornar desencorajador, pois ele não dispõe desse conhecimento, precisando portanto de um treinamento adequado para se familiarizar com o sistema.

Nos últimos anos, a disponibilidade de recursos visuais (gráficos) a um custo baixo tem contribuído muito para que eles sejam usados por um número cada vez maior de interfaces. As vantagens de se utilizar imagem no lugar de texto tornaram-se particularmente relevantes na interação homem-máquina. A abordagem visual atrai a atenção do usuário e deixa-o mais a vontade com relação ao uso dos recursos disponíveis no sistema, pois permite uma interação imediata, sem conhecimento prévio da linguagem. Uma abordagem textual, envolve um maior esforço por parte do usuário, devido a necessidade de saber a sintaxe/semântica dos comandos e/ou linguagens para realizar as ações desejadas.

Uma linguagem textual de consulta, por exemplo SQL, deve no entanto, estar sempre agregada à linguagem visual de consulta de forma transparente, ou seja, servindo como uma ponte de conexão entre a aplicação e os dados, pois é a partir dela que os bancos de dados são acessados, alterados e/ou criados.

Portanto, esta dissertação tem como foco principal a especificação e implementação dessa ponte (camada) de acesso aos dados para um ambiente visual de consultas temporais a banco de dados históricos desenvolvido por Fernandes Silva [24], denominado TVQE (*Temporal Visual Query Environment*). Além disso, foram realizadas modificações e extensões necessárias ao ambiente.

No TVQE, o usuário formula sua consulta a partir da visualização diagramática do esquema do banco de dados, selecionando as classes e atributos necessários através

de uma representação visual, sem se preocupar em conhecer uma linguagem textual de consulta, no caso SQL, para obter os dados necessários.

A seguir descreveremos a motivação, contribuição e objetivos do nosso trabalho, bem como a estrutura de apresentação dos capítulos.

1.1. Motivação

No trabalho desenvolvido por Fernandes Silva [24] foi formalizado o modelo de dados utilizado pelo TVQE e representado visualmente, de modo que o usuário pudesse construir suas consultas. Mas o usuário apenas formulava suas consultas e não obtinha o resultado extensional das mesmas, ou seja, os dados. Assim, especificamos e implementamos um componente, denominado **Tradutor**, cujo objetivo é transformar as consultas realizadas no TVQE em instruções SQL, obtendo portanto seus resultados. Além disso, o TVQE não dispunha de um mecanismo visual para obtenção do resultado intensional da consulta, assim geramos o esquema gráfico do resultado da consulta.

Como a interface do ambiente TVQE não apresentava nenhum tipo de ajuda ao usuário durante o processo de formulação da consulta, foram implementadas mensagens de orientação ao usuário.

No TVQE era permitido apenas formulação de consultas simples, ou seja, consultas cujas condições não referenciavam outras consultas, então implementamos um novo tipo de consulta para o ambiente TVQE, denominada de consulta complexa.

Para ser executado, o ambiente TVQE precisa da instalação do ambiente no qual foi implementado, no caso Java. Portanto, para os usuários que não dispõem deste ambiente e de modo a torná-lo mais acessível, migramos o TVQE para uma versão *applet*, que necessita apenas de um *browser* (*Internet Explorer*, *Netscape*) instalado na máquina do usuário para ser executado.

1.2. Contribuição

Com o presente trabalho se tornou possível formular consultas temporais e convencionais, tendo sido gerado o esquema gráfico do resultado da consulta e a apresentação dos dados do resultado da consulta através de uma forma tabular. Sendo assim possível validar o ambiente TVQE. Além do mais, um outro aspecto muito

importante que também deve ser considerado é a portabilidade do Tradutor, pois as consultas transformadas em instruções SQL podem ser executas em qualquer SGBD.

1.3. Objetivos

Os objetivos principais deste trabalho são os seguintes:

- Incluir algumas modificações no TVQE como a sincronização dos componentes gráficos da interface (painéis, botões e ícones) e mensagens de orientação ao usuário no uso da interface;
- Estender o ambiente TVQE de modo que permita ao usuário realizar consultas complexas, ou seja, consultas que têm outras consultas como parte de suas condições;
- Gerar o esquema gráfico do resultado da consulta (*query schema*) para o ambiente TVQE, onde apenas os nodos selecionados pelo usuário são visualizados;
- Estender o aplicativo TVQE para uma versão *applet*, de modo que mais usuários possam interagir com o mesmo;
- Especificar e implementar o Tradutor.

1.4. Estrutura da Dissertação

Esta dissertação é constituída de seis capítulos, incluindo esta Introdução. Os capítulos estão organizados da seguinte forma:

- O capítulo 2 apresenta uma introdução a banco de dados temporais e sistemas visuais de consulta;
- O capítulo 3 descreve o ambiente TVQE, as modificações e extensões realizadas no mesmo;
- O capítulo 4 apresenta o Tradutor;
- O capítulo 5 descreve os aspectos de implementação relacionados ao Tradutor e as extensões do TVQE;
- O capítulo 6 relata os resultados alcançados e estabelece as direções de pesquisa para este trabalho.

Capítulo 2 - Uma Introdução a Banco de Dados Temporais e Sistemas Visuais de Consultas

A constante evolução das aplicações que utilizam banco de dados tem demandado novas abordagens para modelagem dessas aplicações. Dentre as principais demandas, a modelagem de aspectos dinâmicos e temporais é fundamental para uma melhor representação da realidade.

O paradigma de orientação a objetos demonstrou ser dos mais adequados para modelagem de aspectos dinâmicos, pois estes fazem parte dos objetos e, portanto, estão integrados à estrutura dos dados. Informações temporais, como datas, intervalos de tempo, duração de atividades e restrições temporais, também se fazem presente na maioria das aplicações e precisam ser convenientemente representadas.

Um banco de dados (BD) mantém dados sobre uma organização e suas atividades. Os **bancos de dados convencionais** são projetados para capturar os dados mais recentes, ou seja, os dados correntes. O processo de alteração dos dados em BD é feito adicionando novos dados sobre os já existentes, isto é, através de uma sobreposição parcial ou total dos dados, e isto leva à perda dos dados anteriores à modificação. Portanto, os banco de dados convencionais servem apenas para aquelas aplicações que necessitam de informações sobre os dados atuais do BD, sendo insuficientes para aquelas que requerem informações sobre o passado e/ou futuro dos dados. Enquanto que os **bancos de dados temporais** mantêm os dados passados, presentes e futuros [2], suprindo assim as “insuficiências” dos BDs convencionais.

Este capítulo tem como objetivo introduzir banco de dados temporais, considerando os aspectos de modelagem do tempo e a dimensão tempo. Além de

descrever um pouco sobre o modelo de dados temporal (TRM) e os sistemas visuais de consultas a banco de dados (SVCs), pois todos esses conceitos são necessários para um melhor entendimento dos próximos capítulos deste trabalho.

2.1. Aspectos de Modelagem do Tempo

O tempo pode ser modelado considerando os seguintes modelos: linear, ramificado, cíclico, discreto, denso e contínuo [2].

No modelo **linear**, o tempo evolui passo a passo (linearmente) do passado para o futuro. Isto implica em uma ordenação total entre quaisquer dois pontos no tempo. No modelo **ramificado**, também denominado de hipotético, o tempo é linear do passado até o tempo presente e, a partir deste ponto, o tempo pode se dividir em várias linhas de tempo, cada uma representando um futuro possível descrito por uma seqüência de eventos. A estrutura do tempo ramificado é na realidade uma árvore cuja raiz representa o tempo presente. No modelo **cíclico**, o tempo é associado a processos periódicos, ou seja, ocorrência de ciclos. Por exemplo, uma semana, onde após sete dias, o mesmo dia volta a ocorrer. Quanto à densidade de espaço de instantes temporais, no modelo **discreto** de tempo, este espaço é isomorfo aos números naturais, ou seja, cada ponto no tempo (instante) tem um sucessor. O modelo **denso** de tempo é isomorfo aos números racionais, ou seja, entre quaisquer dois instantes de tempo, sempre existe um outro instante. O modelo **contínuo** de tempo é isomorfo aos números reais, isto é, denso e diferentes dos números racionais, pois não possui espaços e a cada número real corresponde a um instante.

No modelo contínuo, cada número real corresponde a um ponto no tempo (instante) e no modelo discreto, cada número natural corresponde a uma unidade de tempo indivisível, denominada *chronon*. Um *chronon* é uma pequena duração de tempo, representando um segmento de linha na linha do tempo e não um ponto no tempo.

2.2. Tipos de Dados Temporais

O tipo de dado temporal mais básico é o **instante**, que é um *chronon* particular na linha de tempo. Um **evento** é um fato instantâneo, isto é, algo ocorrido em um instante. Podemos citar alguns tipos que representam instante na linguagem SQL, como: DATE, TIME e TIMESTAMP. Existe ainda mais quatro tipo de dados temporais: **período**, **intervalo**, **conjunto de instantes** e **elementos temporais**, descritos a seguir.

Um **período** de tempo representa o tempo entre dois instantes definidos como início e fim. O tipo período não existe em SQL-92, mas já existe em SQL3 [18].

Um **intervalo** de tempo representa uma duração de tempo determinada por um número de *chronons*, que é, uma quantidade de tempo cujo tamanho é conhecido, mas sem nenhum instante inicial ou final específico, por exemplo, três meses. SQL-92 suporta dois tipos de intervalos, mês-ano e segundo-dia.

O dois últimos tipos de dados temporais são **conjunto de instantes**, cujo próprio nome define-o e **elementos temporais** que representam uniões finitas de períodos.

2.3. A Dimensão Tempo

No contexto de banco de dados [2] existem duas dimensões de tempo que são de interesse geral: a dimensão de **tempo válido** e a de **tempo de transação**.

O **tempo válido** é o tempo em que um fato é verdadeiro na realidade modelada. O tempo válido de um evento é o tempo em que o mesmo ocorreu no mundo real, independente do registro deste evento no banco de dados.

O **tempo de transação** é o tempo em que um fato é armazenado no banco de dados. O tempo de transação de um fato identifica o instante da transação que inseriu o fato no BD e a transação que removeu este fato do BD.

Estes tempos podem ser instantes, períodos ou elementos temporais. Se o tempo válido de um fato é um instante, este fato é denominado de evento. O tempo de transação em princípio sempre é um período mas, em muitas aplicações, só é registrado o instante de início da transação.

Esta duas dimensões são ortogonais. Então podemos classificar os bancos de dados temporais de acordo com o tipo de dimensão que cada um suporta:

Banco de Dados de Tempo Válido, suporta o tempo válido;

Banco de Dados de Tempo Transação, suporta o tempo de transação;

Banco de Dados Bitemporal, suporta tanto o tempo válido quanto o tempo de transação;

Banco de Dados Atuais *(Snapshots)*, não suporta nenhuma das dimensões acima citadas, pois considera todos os fatos valendo só no instante atual.

O tempo tem sido adicionado a vários modelos de dados: modelos semânticos, modelos baseado em conhecimento e modelos de dados dedutivos. Contudo, a maioria dos trabalhos de banco de dados temporais é baseado nos modelos relacionais e orientados a objetos.

A tabela 2.1 lista alguns dos modelos de dados relacionais temporais [2] mais utilizados na literatura. Alguns modelos são definidos unicamente sobre o tempo válido ou tempo transação e outros sobre ambos. A tabela 2.2 ilustra alguns dos modelos de dados temporais orientados a objetos [2].

Modelo de Dados	**Dimensão Temporal**	**Identificador**
Modelo de Dados Conceitual Bitemporal	Bitemporal	*BCDM*
Modelo Relacional Temporal	Bitemporal	*Ben-Zvi*
Modelo Relacional Temporal	Válido	*TRM*
Modelo de Dados Relacional Histórico	Válido	*HRDM*
DATA	Transação	*DATA*

Tabela 2.1 – Modelos de dados relacionais temporais

Modelo de Dados	**Dimensão Temporal**	**Identificador**	**Representação do *Timestamp* de Transação**
MATISSE	Transação	*MATISSE*	*Chronon*
Postgres	Transação	*Postgres*	Período
Modelo de Dados Temporal Orientado a Objetos	Bitemporal	*TOODM*	Elemento Temporal
TMAD	Válido	*TMAD*	Não Aplicado
TOM	Bitemporal	*TOM*	Elemento Temporal

Tabela 2.2 – Modelos de dados temporais orientados a objetos

Os modelos de dados temporais podem ser comparados considerando a dimensão de tempo válido através de duas perguntas básicas: como o tempo válido é representado? e como os fatos estão associados ao tempo válido? A tabela 2.3 categoriza a maioria destes modelos ao longo desses dois aspectos [2].

O tempo válido pode ser representado e associado de diferentes maneiras. O tempo válido pode ser representado com identificadores únicos de *chronon* (i.e., instante), períodos ou com elementos de tempo-válido que são conjuntos finitos de períodos. O tempo válido pode ser associado com valores individuais de atributos, grupos de atributos, ou com uma tupla ou objeto inteiro e, finalmente, as restrições sobre os inteiros (ou reais) podem ser usadas para expressar o tempo em que uma tupla é válida. Outras alternativas, tais como associar o tempo válido a conjunto de tuplas (i.e., relações) ou a um grafo de objetos (i.e., um conjunto de objetos, com um atributo de um objeto referenciando um outro objeto no conjunto, formando um grafo conectado), não foram incorporados dentro de nenhum modelo de dados proposto, principalmente porque eles introduzem uma redundância de dados considerável.

	Único *Chronon*	**Período**	**Elementos de Tempo-Válido**
Valores de Atributos com Tempo (*timestamps*)	*ADM* *Caruso* *Lorentzos*	*Bassiouni* *Gardia-2* *McKenzie* *Tansel*	*Bhargava* *Gardia-1* *HRDM* *TOODM* *TOM*
Grupos de Atributos com Tempo	*Sciore-2*		
Tuplas com Atributos de Tempo	*Ariav* *EDM* *HDM* *Lum* *Sadeghi* *Segev* *Wiederhold*	*Ahn* *Ben-Zvi* *Jones* *Navathe* *Sarda* *Snodgrass* *Yan*	*BCDM*
Objetos com Atributos de Tempo	*TEDM*	*OSAM*/T* *TMAD*	*TOM*

Tabela 2.3 – O tempo válido nos modelos de dados temporais

Geralmente, a representação do tempo transação envolve os mesmos resultados do tempo válido, mas existe outras alternativas, descritas abaixo, em que o tempo de transação pode ser representado. A tabela 2.4 ilustra as escolhas feitas nos vários modelos considerando estas alternativas [2].

- O atributo de tempo (*timestamp*) de transação pode ser um único *chronon;*
- O atributo de tempo pode ser um período. Uma tupla recentemente inserida pode ser associada com o período, cujo início represente o momento presente e o fim

um valor especial, denominado *U.C (until changed)*, ou seja, até que haja uma mudança no valor da tupla;

- O atributo de tempo pode ser constituído de três *chronons*;
- O atributo de tempo pode ser um elemento de tempo-transação, que é um conjunto de períodos.

	Único *Chronon*	**Período**	**Três *Chronons***	**Elementos de Tempo-Transação**	**Outros**
Valores de Atributos com Tempo	*Caruso*	*TOM*		*Bhargava TOODM*	*Sciore-1*
Grupos de Atributos com Tempo	*Sciore-2*				*OVM*
Tuplas com Atributos de Tempo	*Ariav DATA DM/T EDM Lomet*	*Postgres Snodgrass Yan*	*Ben-Zvi*	*BCDM*	
Objetos com Atributos de Tempo	*IRIS TIGUKAT*	*TOM*			*IRIS Kim*
Conjunto de Tuplas com Atributos de Tempo	*ADM Ahn*	*McKenzie*			
Grafo de Objeto	*MATISSE TIGUKAT*				
Esquema com Atributos de Tempo	*MATISSE TIGUKAT*	*McKenzie Postgres*		*BCDM*	*MATISSE*

Tabela 2.4 – O tempo de transação nos modelos de dados temporais

A seguir descreveremos os conceitos de sincronismo e dependência temporal existentes entre atributos de uma relação de tempo válido, e a definição de normalização temporal, dentro do contexto do modelo relacional temporal.

2.4. Modelo Relacional Temporal (*Temporal Relational Model* - TRM)

O TRM incorpora a semântica temporal do mundo real no modelo relacional [19]. A seguir, definimos os conceitos para modelagem de banco de dados temporal de relações **estáticas** e de **tempo-variado**; considerando a consistência dessas relações, a classificação de atributos temporais e a noção de normalização temporal.

Segundo Navathe et al [19], um banco dados temporal é definido como uma união de dois conjuntos de relações R_s e R_t, onde R_s é o conjunto de todas as relações estáticas, ou seja, relações a cujas tuplas não estão associadas valores de atributos temporais (*timestamps*), e R_t o conjunto de todas as relações variantes no tempo (*time-varying relations* - *TVR*). Neste modelo, toda relação variante no tempo tem dois atributos temporais obrigatórios: T_S (tempo-de-início) e T_E (tempo-de-fim). Estes atributos temporais correspondem ao início e o fim de um período de tempo, respectivamente. Em uma *TVR*, o valor de um atributo pertencente a um tupla está associado com os *timestamps* se for continuamente válido no período $[T_S, T_E]$. As relações variantes no tempo são também chamadas de relações de tempo-válido.

Considere a relação *Empregado* ilustrada na tabela 2.5. A tupla $< e, s, p, t_1, t_2 >$ nesta relação significa que um empregado *e* ganha um salário *s*, e que a este fato está relacionado o tempo t_1 e t_2; isto é, o salário *s* de *e* é continuamente válido no período $[t_1,t_2]$.

O sincronismo e a dependência temporal existente entre os atributos de uma relação de tempo-válido são fatores relevantes para realizar a normalização temporal. Portanto, a seguir definimos esses conceitos a fim de que possamos entender a normalização temporal que faz parte do processo de criação do esquema do BD temporal.

2.4.1. Sincronismo e Dependência Temporal

Um conjunto de atributos que variam com o tempo (*time-varying attribute – TVA*) em uma relação é chamado de **síncrono** se a todos eles podem ser uniformemente associados e diretamente aplicados os mesmos valores temporais em cada tupla da relação [19].

Considere, por exemplo, a relação *Empregado* ilustrada na tabela 2.5. Aqui, um empregado tem um aumento salarial se e somente se tiver uma promoção. Assim, os atributos *Salário* e *Posição* formam um conjunto de atributos síncronos. Os atributos T_S e T_E formam o conjunto de atributos temporais presentes na relação de tempo-válido *Empregado*, onde T_S indica o início e T_E o fim de um período de tempo ao qual toda tupla da relação está associada.

EmpNum	Salário	Posição	T_S	T_E
33	20K	Datilógrafa	12	24
33	25K	Secretária	25	35
45	27K	Engenheiro Júnior	28	37
45	30K	Engenheiro Senior	38	42

Tabela 2.5 – Relação Empregado

Considere uma outra relação chamada de *Manutenção* ilustrada na tabela 2.6, onde o atributo *#Avião* é o identificador de uma aeronave, *Parte* é a parte de uma aeronave, *Problema* descreve a natureza do problema de uma parte da aeronave, *Lugar* é o nome do lugar onde o avião é consertado e *Custo* é o custo associado a manutenção da aeronave, T_S é o tempo quando um evento particular de manutenção ocorre e T_E é o tempo quando termina.

#Avião	Parte	Problema	Lugar	Custo	T_S	T_E
91	Roda	Separada	Atlanta	1000	10	20
105	Porta	Quebrada	Nova Iorque	2000	35	47
105	Porta	Desengonçada	Los Angeles	2500	55	62
142	Asa	Rachada	Boston	7000	60	72

Tabela 2.6 – Relação Manutenção

Neste exemplo, um quase sincronismo é imposto em todos os atributos que variam com o tempo da relação pela gravação de seus valores associados com o mesmo par de *timestamps* (T_S e T_E).

O sincronismo de atributos que variam com o tempo ocorre em dois contextos: o **sincronismo atual (válido)**, que é definido pelo comportamento natural, e o **quasi (físico) sincronismo**, que é imposto pela simultaneidade da gravação coletiva destes valores de atributos [19].

A noção de dependência temporal é algo análogo à dependência multivalorada. A **dependência multivalorada** acontece quando dois ou mais fatos não relacionados, cada um envolvido em um-para-muitos (1:N) relacionamentos, são colocados em uma única relação. A **dependência temporal** ocorre quando dois ou mais fatos temporalmente não relacionados são envolvidos em uma relação variante no tempo.

2.4.2. Normalização Temporal

Uma relação está na **forma normal temporal** (*Temporal Normal Form - TNF*), segundo Navathe et al [19], se e somente se estiver na forma normal de Boyce-Codd[1] e não existir dependência temporal entre os atributos não chaves da relação.

Sempre é possível decompor uma relação, se existe uma dependência temporal, em duas ou mais relações temporalmente normalizadas através de uma partição apropriada de atributos e uma combinação dos intervalos de tempo relevantes. Esta decomposição satisfaz a propriedade que não permite dois ou mais atributos temporalmente dependentes em um mesma relação. Em geral, o conjunto de atributos que variam com o tempo em uma relação podem ser particionados dentro de um número mínimo de conjuntos até que tenha apenas um único atributo temporalmente dependente dentro de cada subconjunto.

Por exemplo, a relação *Salario-Gerente* (tabela 2.7) pode ser decomposta em duas relações, *Gerente* (tabela 2.8) e *Salario* (tabela 2.9). Note que estas decomposições não acarretaram perdas na semântica da relação *Salario-Gerente*, apenas representam a informação sobre o tempo de vida de um atributo de uma maneira compacta e concisa. A normalização temporal assegura que o tempo de vida de uma tupla e seus valores de atributos sejam os mesmos após a decomposição. O valor *Now* para o atributo T_E indica o instante presente.

[1] Uma relação está na forma normal de Boyce-Codd (*BCNF – Boyce-Codd Normal Form*) se e somente se, cada **determinante** existente na relação for uma chave candidata, onde um **determinante** é um atributo do qual algum outro atributo é funcionalmente dependente [32].

EmpNum	Salario	Gerente	T_S	T_E
52	18K	João	5	9
52	20K	João	10	20
52	25K	João	21	29
52	25K	José	30	38
52	31K	José	39	42
52	31K	João	43	47
52	38K	João	48	*Now*
97	30K	Marcos	12	17
97	35K	Marcos	18	*Now*

Tabela 2.7 – Relação Salario-Gerente

EmpNum	Gerente	T_S	T_E
52	João	5	29
52	José	30	42
52	João	43	*Now*
97	Marcos	12	*Now*

Tabela 2.8 – Relação Gerente

EmpNum	Salario	T_S	T_E
52	18K	5	9
52	20K	10	20
52	25K	21	38
52	31K	39	47
52	38K	48	*Now*
97	30K	12	17
97	35K	18	*Now*

Tabela 2.9 – Relação Salario

2.4.3. A Necessidade da Normalização Temporal

A idéia por trás da normalização temporal é que as tuplas são semanticamente independentes uma das outras. Em uma relação de tempo-válido não normalizada, toda tupla tem informação incompleta sobre o tempo de vida de seus atributos, tornando-se então, semanticamente dependente de outras tuplas para a determinação de tal informação.

Considere a relação *Salario-Gerente* (tabela 2.7) que tem a dependência temporal, *Salario* $\leftarrow T \rightarrow$ *Gerente*. A tupla < 52, 20K, João, 10, 20 > tem T_S=10 e T_E=20, isto não representa o período de tempo em que João é gerente, neste caso, se refere ao período de tempo em que o empregado 52 tem salário igual 20K. Então, esta

tupla tem informação incompleta relativa ao tempo de vida em que João foi gerente. Portanto, uma mudança assíncrona no valor de um atributo divide a informação do tempo de vida de outros atributos sobre tuplas diferentes.

Considere, por exemplo, a seguinte consulta: *Quando João tornou-se gerente do empregado 52?*, aplicada a relação não normalizada *Salario-Gerente* (tabela 2.7). O resultado incorreto desta consulta está ilustrado na tabela 2.10.

Nas relações normalizadas temporalmente a recuperação é simples porque a informação sobre o tempo de vida dos atributos aparece de uma forma não fragmentada. A mesma consulta, se aplicada a relação normalizada temporalmente *Gerente* (tabela 2.8), recuperaria o resultado correto da consulta, ilustrado na tabela 2.11.

EmpNum	**Gerente**	**T_S**	**T_E**
52	João	5	9
52	João	10	20
52	João	21	29
52	João	43	47
52	João	48	*Now*

Tabela 2.10 – Resultado incorreto da consulta

EmpNum	**Gerente**	**T_S**	**T_E**
52	João	5	29
52	João	43	*Now*

Tabela 2.11 – Resultado correto da consulta

Portanto, as relações que não estão na TNF tem o tempo de vida de um atributo fragmentado sob várias tuplas, todas as consultas que envolvem operadores temporais de comparação[2] podem recuperar resultados incorretos. Como os atributos que variam com o tempo podem mudar de maneiras completamente diferentes, então, haveria uma repetição redundante dos valores de um atributo cuja variação fosse maior que a dos outros atributos pertencentes a relação. Logo, a normalização temporal evita estas redundâncias e anomalias de alteração e recuperação.

[2] Estes operadores servem para comparar dois períodos de tempo diferentes, por exemplo, os operadores *follow*, *during*, *before*, *after*, entre outros serão descritos no próximo capítulo.

2.5. Linguagens de Consultas Temporais

As linguagens de consultas temporais foram propostas baseadas em seus modelos de dados e nas linguagens convencionais de consultas dos modelos [2]. Assim, surgiram várias linguagens de consultas temporais relacionais e orientadas a objetos, por exemplo, TSQL2 e TOSQL que baseiam-se nas linguagem SQL-92 e SQL, respectivamente.

Uma das grandes dificuldades nos modelos de dados temporais, é que não existem Sistemas Gerenciadores de Banco de Dados (SGBDs) no mercado que dêem suporte a uma linguagem de consulta temporal. Os SGBDs comerciais apenas suportam alguns tipos de dados temporais como: DATE, TIME, TIMESTAMP e INTERVAL. O tipo período, apesar de ser muito utilizado, não é suportado diretamente por nenhum SGBD comercial que utiliza o padrão SQL-92, uma das razões para que isto ocorra, é a sua relativa facilidade de ser simulado através de outros tipos de dados temporais, mas já foi incluído no padrão SQL3 [18].

A evolução da linguagem SQL considerando seus aspectos temporais deu-se da seguinte forma: o primeiro padrão surgido foi o SQL-86 que não suportava nenhum tipo de dado temporal, apesar de alguns SGBDs comerciais a partir de 1980 suportarem o tipo DATA. No SQL-89 foi adicionado o suporte à integridade referencial, mas ainda não tinha nenhum tipo de dado temporal. Vários tipos de dados temporais foram introduzidos no SQL-92: DATE, TIME, TIMESTAMP e INTERVAL. Agora, temos o padrão SQL3 cuja parte temporal relacionada com tipos de dados temporais inclui um novo construtor chamado PERIOD, onde podem ser especificados dados do tipo período a partir de tipos de dados temporais como *date, time, timestamp* e de tipos numéricos, por exemplo: *period(date), period(numeric)* [18].

A seguir discutiremos um pouco sobre os Sistemas Visuais de Consultas (SVCs) a banco de dados, pois o nosso trabalho tem como base um SVC para consultar dados históricos. Portanto, precisamos conhecer alguns conceitos como por exemplo, o que é um SVC e as evoluções de suas interfaces visuais de consultas.

2.6. Sistemas Visuais de Consultas - SVCs

Os SVCs podem ser definidos como sistemas de consultas baseados em representações visuais e incluem uma linguagem para expressar consultas em um formalismo visual, VQL (*Visual Query Language*) [24]. Elas são uma alternativa às linguagens textuais de consulta e são orientadas a um grande grupo de usuários, os usuários inexperientes (casuais), que se caracterizam por desconhecerem ou ignorarem a estrutura interna do banco de dados e sua linguagem de consulta, precisando portanto de outros meios para acessar os dados do banco de dados.

Vários SVCs gráficos têm sido propostos, mas apenas poucos deles apresentam uma definição formal. Todos esses sistemas baseiam-se principalmente na idéia de propor uma nova representação visual para modelos clássicos (relacional) e não-clássicos (objeto-relacional, TOM [25]) de banco de dados, cujo mecanismo de interação baseia-se no paradigma da manipulação direta[3] [1]. A seguir daremos uma breve descrição da evolução das interfaces visuais para banco de dados.

2.7. Evolução das Interfaces Visuais para Banco de Dados

As interfaces visuais têm evoluído consideravelmente nos últimos anos. Mostraremos a seguir os SVCs abordando a evolução das principais representações visuais: tabular, diagramática, icônica e a híbrida/multimodal.

- Interfaces Tabulares

Na representação tabular, os dados são organizados e visualmente representados como tabelas e os relacionamentos entre os dados como uma justaposição de retângulos. Esta representação é bastante adequada para a estrutura do modelo relacional. A primeira interface tabular conhecida mundialmente foi a *QBE (Query-By-Example).* Na interface *QBE* as tabelas são mostradas visualmente, sobre as quais o usuário formula

[3] É um estilo de interação, onde para realizar uma tarefa, o usuário seleciona um objeto e em seguida realiza uma ação sobre o objeto.

sua consulta preenchendo os atributos da tabela com exemplos e restrições necessárias para o resultado da consulta [24].

Na década de 90, mesmo com o paradigma de orientação a objeto integrado a banco de dados, novas interfaces tabulares surgiram, devido à simplicidade da representação tabular e do modelo relacional. A estas novas interfaces foram incorporadas novas características que possam satisfazer a demanda das novas aplicações, como a interface *GRADI*, que foi desenvolvida para um banco de dados relacional multimídia. Outras interfaces tabulares surgiram [24], como a *FormDoc*, *TableTalk* e *HIBROWSE*.

- Interfaces Diagramáticas

Diagramas em um SVC representam visualmente entidades e relacionamentos entre entidades, de uma forma similar a estrutura de um grafo. Sua popularidade surgiu a partir da representação visual dos modelos semânticos, em particular o modelo E-R. A primeira interface diagramática surgiu em 75, a interface *Cupid*. Segundo Fernandes Silva [24], a interface *Cupid* foi projetada como uma interface de alto nível para um banco de dados relacional e utiliza símbolos para representar componentes de uma consulta. O usuário seleciona componentes relevantes e constrói sua consulta a partir destes componentes.

Podemos citar outros exemplos de interfaces diagramáticas, tais como, *GOOD*, ConTOM [20], *AMAZE*, entre outras [24]. Contudo, os diagramas conseguem captar a visualização global dos dados, mostrando o inter-relacionamento entre os mesmos. Adicionalmente, a maioria das interfaces diagramáticas exige que o usuário manipule a estrutura do esquema conceitual. Contudo, o processo de criação da estrutura do esquema conceitual é feito pelo projetista do banco de dados, cuja visão sobre os dados é diversa da visão do usuário final.

- Interfaces Icônicas

Os ícones proporcionam uma representação visual de um conceito ou de uma função. Os ícones em um SVC representam tanto as entidades do mundo real como as funcionalidades disponíveis no sistema. Contudo, a representação icônica não se tornou

popular em banco de dados pelo seguinte motivo: o esquema conceitual do banco de dados não é visualizado explicitamente nos SVCs icônicos, desde que não se consegue representar explicitamente o relacionamento entre as entidades.

A primeira interface icônica para banco de dados foi a *IconicBrowser*. Entretanto existe outras [24]: *Iconographer*, *QBI* e *Oggeto desktop*.

- Interfaces Híbridas / Multimodais

As interfaces híbridas/multimodais suportam aplicações complexas e diversos tipos de usuários, permitindo o acesso aos dados através da combinação de diferentes representações visuais.

Nas interfaces multimodais, os modos de interação são manipulados independentemente, enquanto que, na abordagem híbrida os diferentes modos são integrados. A abordagem híbrida surgiu como uma forma de compensar a deficiência de uma única representação visual em alguns aspectos, por exemplo, ícones relacionados através de conectores como os diagramas, podem representar visualmente um esquema conceitual. Os SVCs híbridos surgiram na década de 80 [24], por exemplo, o *SKI*. Como exemplo de uma interface multimodal, temos a interface do ambiente TVQE, pois combina tanto a representação icônica, através de uma "agenda gráfica", como a diagramática, através do esquema gráfico. O ambiente TVQE será descrito em maiores detalhes no próximo capítulo.

2.8. Conclusão

Neste capítulo introduzimos alguns conceitos necessários para que o nosso trabalho de dissertação seja melhor compreendido. Como o objeto do trabalho é um ambiente visual de consultas temporais, no caso TVQE, viu-se a necessidade de apresentar alguns conceitos, tais como: o que é um banco de dados temporal, aspectos de modelagem do tempo, os tipos de dados temporais existentes, a questão da normalização temporal, pois vai ser utilizada no processo de criação do banco de dados temporal, as linguagens de consultas temporais e, por último, o processo evolutivo das interfaces visuais de consultas.

Capítulo 3 - O Ambiente TVQE

O TVQE é um SVC para banco de dados históricos, o qual inclui um conjunto de primitivas gráficas temporais[4] e possui duas representações visuais do esquema de banco de dados, a representação diagramática (através de grafos) e icônica (explora a metáfora de uma "agenda").

Nesse contexto, este capítulo tem como objetivo apresentar o ambiente TVQE, levando-se em consideração o modelo de dados, características do usuário, apresentação das informações e as modificações e extensões realizadas.

Os exemplos ilustrativos referentes a este capítulo serão baseados em um esquema conceitual de uma agência de empregos com a seguinte funcionalidade: uma agência de empregos presta serviço a seus clientes a procura de empregados para eles. Toda pessoa antes de ser empregada, é cadastrada como candidata a um emprego. Assim que as empresas fornecem certas informações (projetos com equipes que necessitam de novos empregados, etc) às agências, estas analisam os candidatos cadastrados e encaminham os mais qualificados de acordo com os requisitos propostos pela empresa.

[4] As primitivas gráficas temporais consistem de operações gráficas elementares, que podem ser usadas como componentes básicos para expressar visualmente consultas que envolvem seleção e projeção de tempo válido [24].

3.1. Modelo de Dados

Um banco de dados e uma linguagem de consulta são formalmente definidos em termos de um modelo de dados e um conjunto de operadores, respectivamente [28]. Por causa do crescente interesse no campo de interação homem-máquina, o uso da representação do modelo de dados, não só para descrever o esquema, mas principalmente para acessar o banco de dados (BD) tem ganho uma grande importância no ambiente de desenvolvimento de interfaces visuais de consultas.

A partir dessas considerações, Catarci et al [28] propõem um conjunto de primitivas gráficas baseado em um modelo gráfico, chamado de *Graph Model*. As características principais do *graph model* e das primitivas gráficas são:

(a) O *graph model* permite-nos definir um *Graph Model Database* (GMDB)[5] em termos de um tripla ⟨g, c, m⟩, onde ***g*** é um *Typed Graph*, ou seja, uma estrutura orientada a grafo que modela o esquema conceitual da aplicação, ***c*** é um conjunto de restrições de integridade das classes de objetos representados em ***g*** (***g*** e ***c*** juntos representam o nível intensional do BD, ou seja, esquema do BD) e ***m*** é uma interpretação das classes em ***g*** (***m*** representa o nível extensional do BD, ou seja, as instâncias do BD);

(b) A semântica das primitivas gráficas é caracterizada em termos de **transformações de grafos**, de forma que na avaliação de uma consulta, a partir da tripla inicial representando o GMDB, é produzida uma tripla final, contendo exatamente a informação solicitada.

A generalidade da abordagem adotada por [28] permite representar uma grande classes de modelos, tais como: os modelos relacionais, semânticos e orientados a objetos, em termos de construtores do *graph model*. As primitivas gráficas podem ser efetivamente usadas para caracterizar a semântica das linguagens de consultas mais populares definidas tanto nos modelos semânticos quanto nos modelos orientados a objetos. Em princípio, o *graph model* pode ser utilizado como um modelo interno de algum sistema visual de consulta, provendo um formalismo básico para caracterizar a semântica das operações visuais definidas no modelo externo.

[5] É um banco de dados conceitual, utilizado para os usuários realizar suas consultas [24].

O modelo de dados utilizado pelo TVQE é o ***Temporal Graph Model* (TGM)**. O TGM apresenta um formalismo baseado em grafo para representar e consultar banco de dados temporais. Ele representa a extensão temporal do modelo *Graph Model*, onde o *Typed Graph* é estendido para uma representação temporal e primitivas gráficas temporais (*Temporal Graph Primitives - TGPs*) são adicionadas [24].

Um esquema de banco de dados é representado no *Typed Graph* em termos de classes e relacionamentos entre classes (denominado papéis). Uma classe é uma abstração de um conjunto de objetos com características comuns e um relacionamento entre classes representa associações entre objetos da classe. A figura 3.1 ilustra um *Typed Graph*, com extensão temporal, relacionado ao exemplo de uma agência de empregos descrita no início deste capítulo.

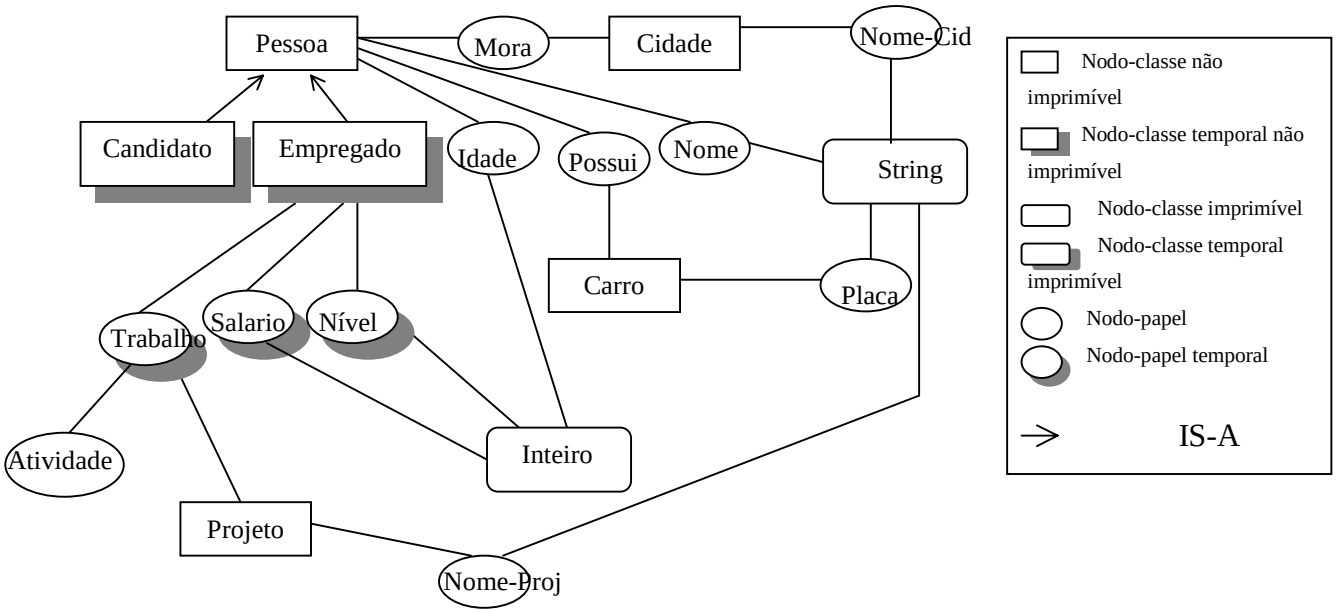


Figura 3.1 – Um *typed graph* g com extensão temporal

Os nodos imprimíveis representam o conjunto de classes cujas instâncias são valores de domínio (ex.: inteiro, *string*, etc) e os nodos não imprimíveis representam o conjunto de classes cujas instâncias são identificadores de objetos (ex.: Pessoa, Empregado).

O TVQE foi desenvolvido para atender as necessidades dos usuários com as seguintes características.

1. Não precisa conhecer o esquema da aplicação;
2. Não precisa aprender uma linguagem textual de consulta, no caso SQL;
3. Ocasionalmente interage com o banco de dados;

4. Conhece o conteúdo do banco de dados;
5. Tem noções sobre banco de dados (classes e relacionamentos em um esquema conceitual);
6. Interações prévias com interfaces de manipulação direta.

A seguir descreveremos como as informações são apresentadas no TVQE considerando o *layout* da interface.

3.2. Apresentação das Informações

As informações são apresentadas no TVQE através de uma janela gráfica constituída de *menus*, ícones, botões, painéis, diagramas e de uma representação visual de uma agenda "gráfica". O *layout* da janela principal é ilustrado na figura 3.2 e contém as seguintes informações:

1. **Área de Menu e Ícones** – Contém um *menu* com botões que representam operações (abrir, fechar e salvar) realizadas sobre um esquema conceitual, além de um botão de saída da aplicação, e um conjunto de ícones que representam as operações realizadas pelo usuário sob suas consultas;
2. **Área de Título** – Contém o título dos painéis da janela principal;
3. **Área de Visualização, denominada Janela de Esquema (*Schema Window*)** - Esta área contém o esquema do banco de dados representado visualmente de três alternativas, tais como: uma árvore de contexto *(context-tree)*, um esquema gráfico *(graph schema)* e um esquema de consulta *(query schema)*;
4. **Área de Interação, denominada Janela de Interação (*Interaction Window*)** - É formada basicamente por dois painéis, um constitui a agenda "gráfica" e o outro é um painel destinado a consultas complexas, que serão descritas em maiores detalhes na seção 3.6.2;
5. **Área de Ajuda** – Esta área contém um painel de mensagens que orienta o usuário no processo de interação com o sistema.

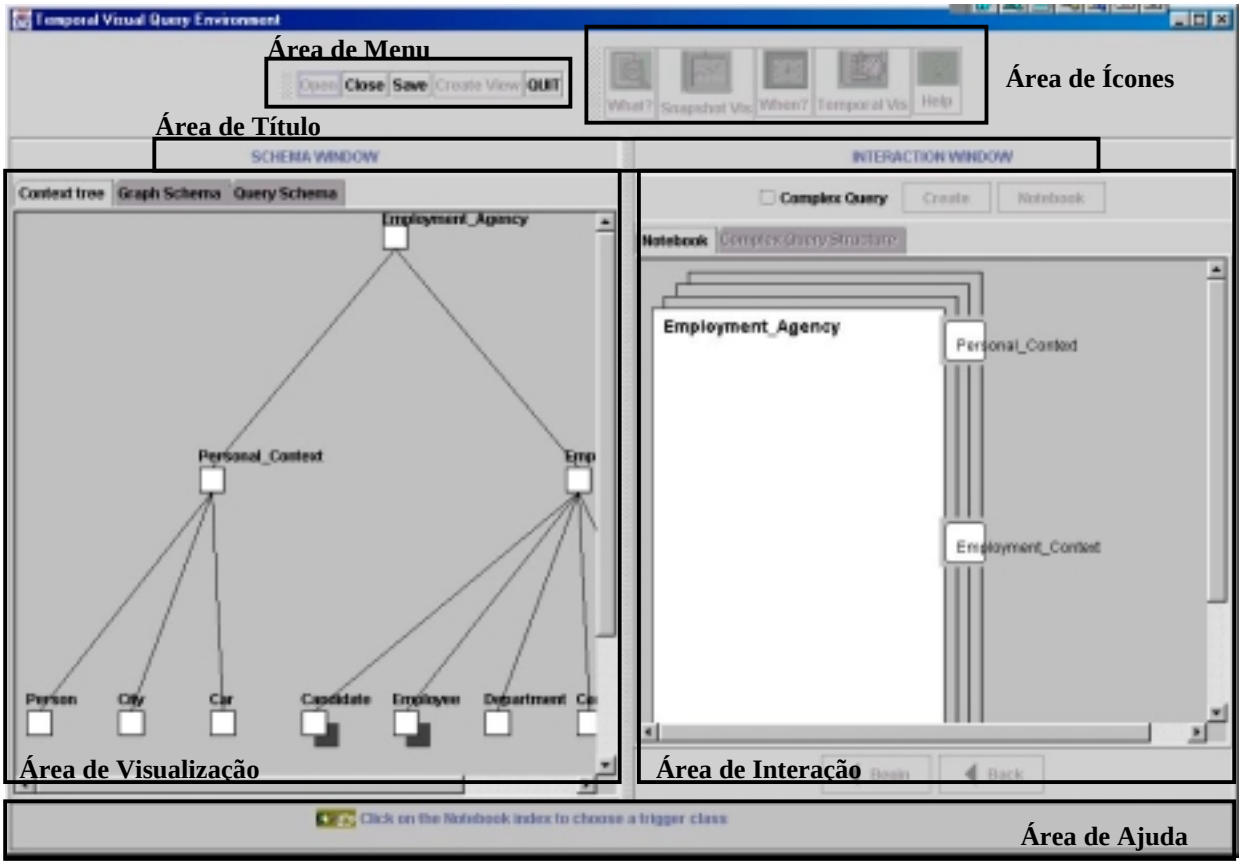


Figura 3.2 – *Layout* da janela principal do TVQE

A seguir descreveremos em maiores detalhes as janelas gráficas (janelas de interação e esquemas) que fazem parte do TVQE.

- Janela de Esquemas (*Schema Window*)

A janela de esquemas visualiza primeiro o esquema conceitual como uma árvore de contextos (*Context Tree*), que é uma estrutura *top-down*, onde um contexto representa um conceito em uma forma abstrata, mas não contém objetos do mundo real como suas instâncias. Sua raiz e os nodos intermediários representam contextos, e as folhas representam classes. Os contextos e as classes são representados visualmente como quadrados e, as classes temporais como quadrados sombreados. A figura 3.3 ilustra um exemplo de uma árvore de contextos no TVQE relacionada a uma agência de empregos.

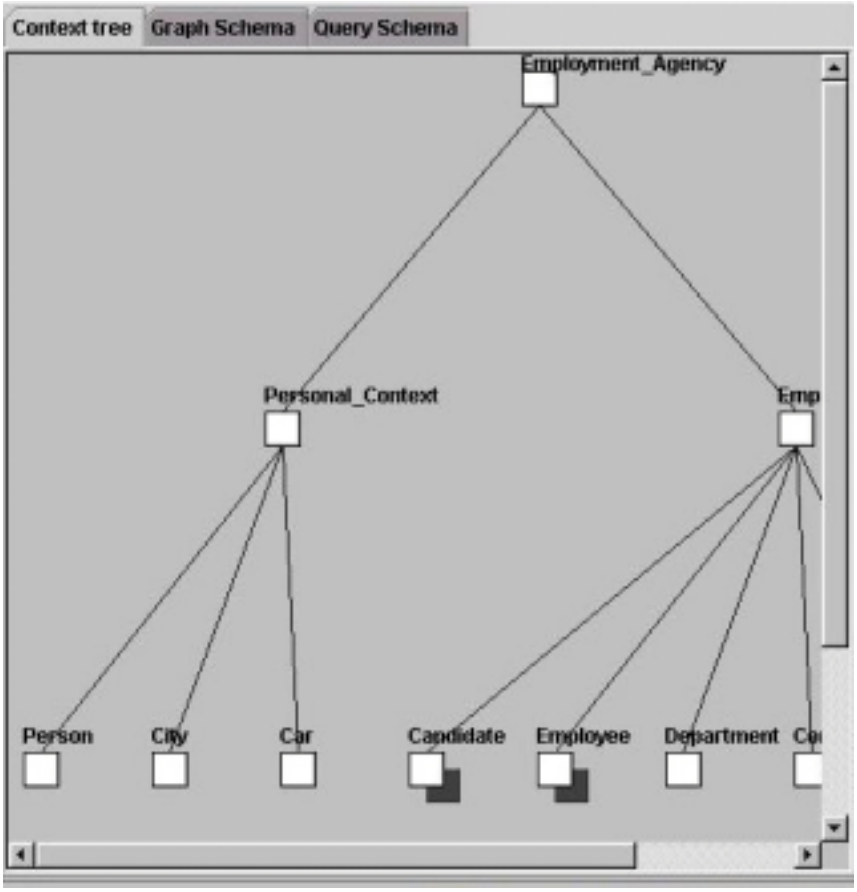


Figura 3.3 – Uma árvore de contextos de uma agência de empregos no TVQE

A segunda forma de representar o esquema conceitual do banco de dados é feita através de um esquema gráfico (*Graph Schema*), que possui quadrados representando classes de objetos (ex.: *Person, Car*) e círculos representando atributos (ex.: *Name, Plate)* e relacionamentos entre as classes (ex.: *Owns, Job)*. Os quadrados e círculos sombreados representam as classes e atributos/relacionamentos temporais, respectivamente. O esquema gráfico é uma versão simplificada do *Typed Graph*, onde os **nodos-papéis** representam atributos de classes e relacionamentos entre classes, os **nodos-classes não-imprimíveis**, representam as classes e os **nodos-classes imprimíveis** foram abstraídos. A figura 3.4 ilustra um esquema gráfico de um agência de empregos no TVQE.

A terceira representação do esquema conceitual é o subgrafo (*Query Schema* - esquema de consulta), como o esquema de consulta faz parte de uma das extensões do ambiente TVQE, então deixamos para discuti-lo melhor na seção 3.5.1.

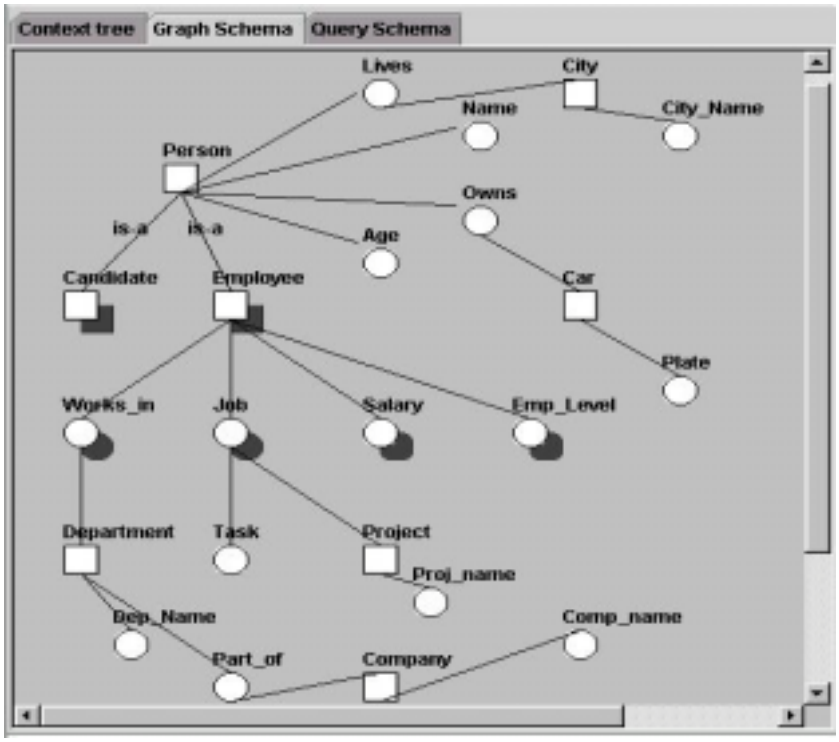


Figura 3.4 – Esquema gráfico de uma agência de empregos no TVQE

A janela, denominada **janela de visualização dos dados**, mostra o resultado da consulta na forma tabular, ou seja, uma tabela é visualizada com os atributos de classe selecionados no processo de formulação da consulta e seus respectivos dados. Esta janela é visualizada quando o usuário aciona os ícones Visualização Corrente *(Snapshot Vis)* ou Visualização Temporal *(Temporal Vis).*

- Janela de Interação (*Interaction Window*)

Através dessa janela, o usuário realiza a maioria dos processos de interação para formulação da consulta, ou seja, a seleção dos índices da agenda "gráfica" que representam classes, atributos e relacionamentos que fazem parte da especificação de uma consulta.

A agenda "gráfica" representa visualmente um esquema de banco de dados. Cada "folha" da agenda representa um contexto ou uma classe. Sempre que uma folha representa um contexto, os "índices" da agenda representam outros contextos ou classes. Caso a folha represente um classe, então os índices representam todas as suas propriedades (relacionamentos e atributos). Os índices da agenda podem assumir quatro diferentes estados, descritos abaixo.

1. **Selecionado** , isto ocorre, quando o índice é selecionado;
2. **Não-Selecionado** , quando o índice não é selecionado;
3. **Visualizado**, quando o índice é selecionado duas vezes seguidas, apenas os índices que representam atributos podem assumir este estado;
4. **Navegado**, quando o índice foi selecionado durante a formulação da consulta.

Estes estados são refletidos visualmente na agenda "gráfica". O estado **selecionado** é representado com o índice pintado de vermelho, o **não-selecionado** o índice não é pintado, o **visualizado** o índice é pintado e contornado de vermelho e o **navegado** o índice é apenas contornado de vermelho. A figura 3.5 ilustra estes quatro estados, onde os índices Salário *(Salary)*, Nome *(Name)*, Idade *(Age)* *e* Mora *(Lives)* estão visualizado, selecionado, não selecionado e navegado, respectivamente. Note que estes estados são refletidos no esquema gráfico (*graph schema*) presente na janela de visualização.

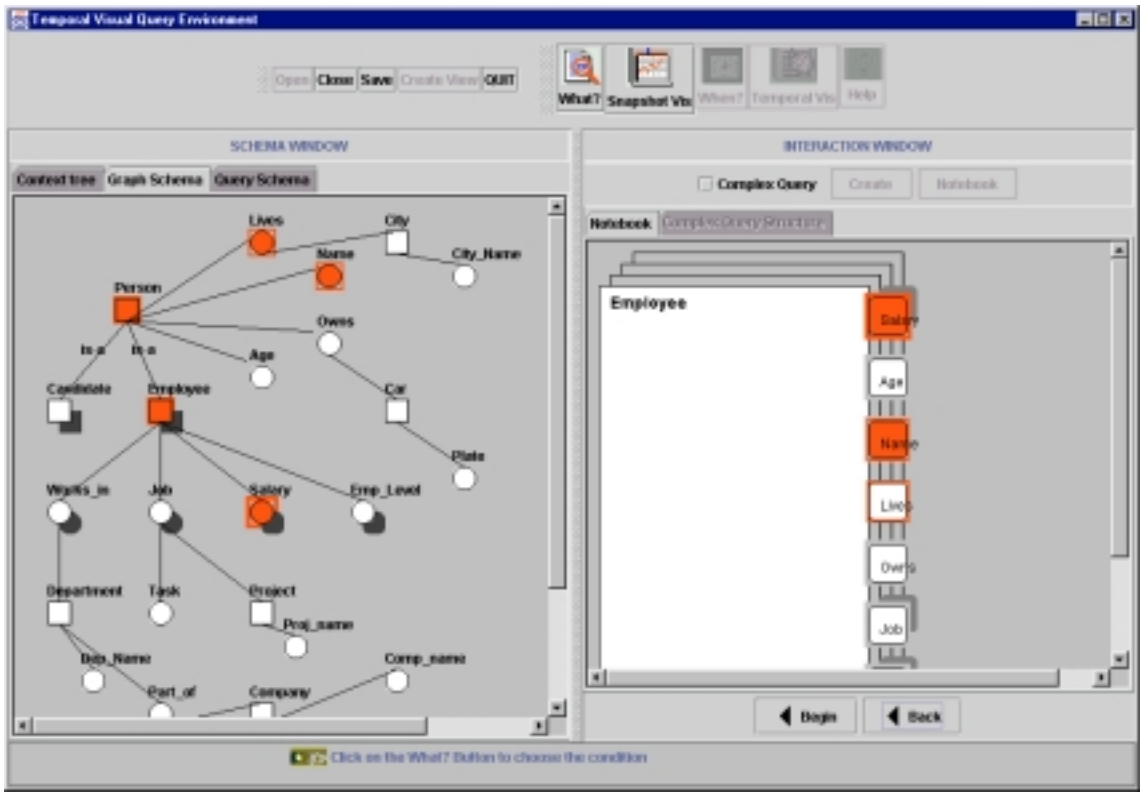


Figura 3.5 – Estados dos índices na agenda gráfica

3.3. Entrada e Saída das Informações da Consulta

A **entrada** das informações para expressar uma consulta é feita através de janelas de diálogos, que são visualizadas quando os ícones *O que? (What?)* e *Quando? (When?)* são acionados. A seguir descreveremos a estrutura (*layout*) dessas janelas.

Quando o ícone *When?* é acionado, isto significa que o usuário deseja expressar alguma condição temporal. A figura 3.6 ilustra o *layout* da janela *When?* quando selecionamos o índice na agenda que representa a classe temporal empregado *(employee)*. A seguir descreveremos o *layout* desta janela.

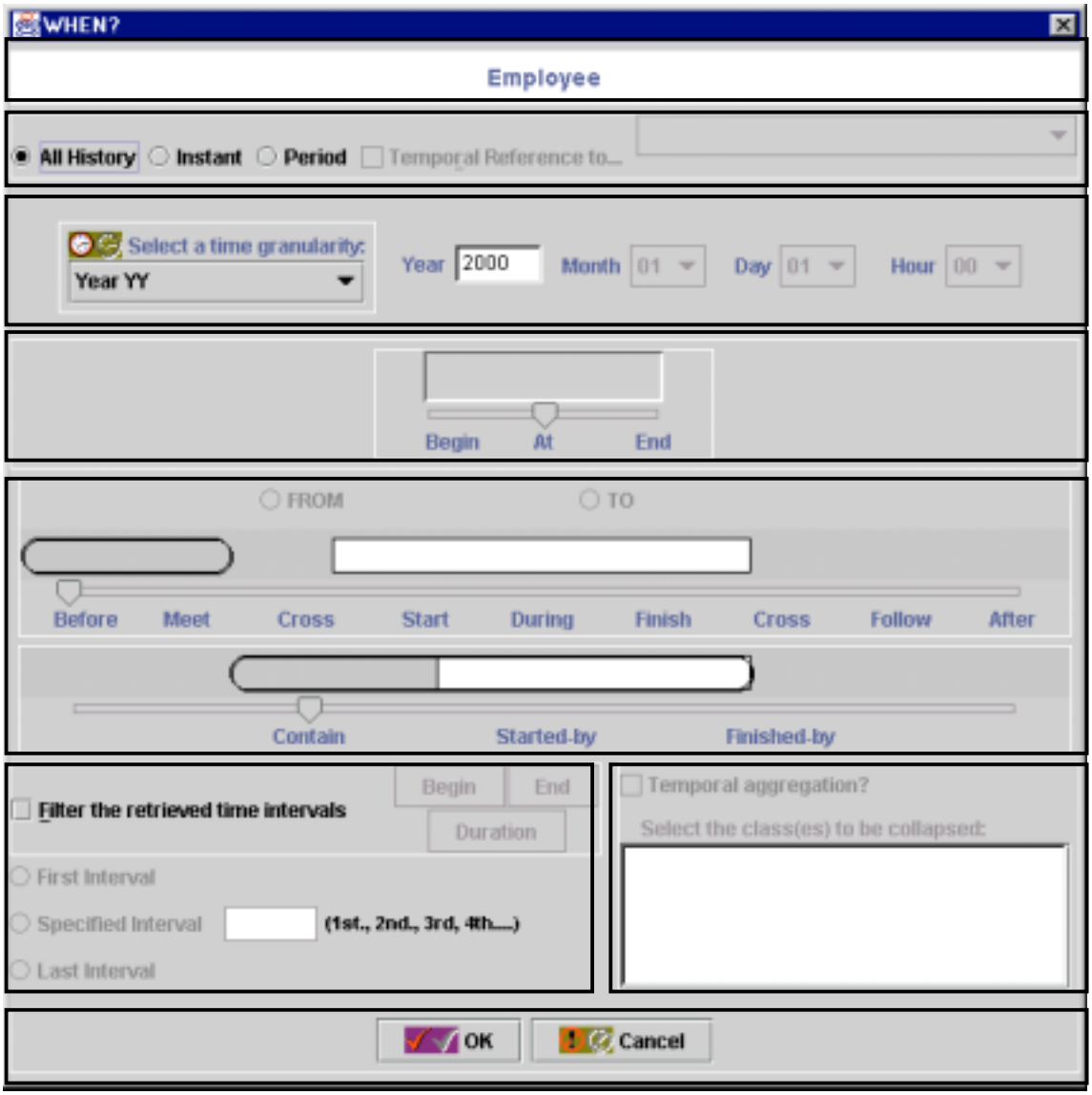


Figura 3.6 - *Layout* do painel *When?*

1. **Painel Visualizador (Painel 1) -** Este painel mostra o índice da agenda selecionado pelo usuário, ao qual será aplicado à condição. Este índice representa uma classe, atributo ou relacionamento temporal;
2. **Painel de Opções da Condição Temporal (Painel 2) -** Este painel é composto pelas seguintes condições temporais: *All History*, que Recupera toda a história de uma classe, relacionamento ou atributo temporal, *Instant* e *Period*, que recuperam um fato instantâneo e histórico, respectivamente e *Temporal Reference to...*, que faz referência a outra classe, atributo ou relacionamento temporal;
3. **Painel de Granularidades Temporais (Painel 3) -** Este painel contém as granularidades temporais (ano, mês, dia e hora) no qual o usuário especifica uma constante temporal dependendo da granularidade escolhida;
4. **Painel para especificar um predicado sobre um instante de tempo (Painel 4) -** Este painel utiliza operadores temporais que serão aplicados sobre um instante de tempo ***t***. Portanto, temos os seguintes operadores: *begin* e *end* que recuperam as instâncias cujo tempo de vida inicia e termina em ***t***, respectivamente, e o operador *at* que recupera as instâncias cujo tempo ***t*** deve estar dentro intervalo das instâncias;
5. **Painel para especificar um predicado sobre dois período de tempo (Painel 5) -** Este painel utiliza operadores temporais que serão aplicados aos períodos da consulta e do banco de dados. Seja ***J*** o período de tempo da consulta e ***I*** o período de tempo presente no banco de dados, então temos os seguintes operadores temporais que relacionam estes dois períodos: *before(I, J), meet(I, J), cross(I, J), start(I, J), during(I, J), finish(I, J), cross(J, I), follow(I, J), after(I, J), contain(I, J), started-by(I, J)* e *finished-by(I, J)*;
6. **Painel de Filtragem (Painel 6) -** Este painel é acionado sempre quando é escolhida a condição temporal *all history*, pois esta condição recupera todo o histórico dos dados relacionados à classe, relacionamento ou atributo temporal selecionado pelo usuário. Os dados recuperados podem ser filtrados através dos operadores *First Interval* e *Last Interval* que recuperam o primeiro e último intervalo, respectivamente e pelo operador *Specified Interval* que recupera um intervalo qualquer informado pelo usuário;

7. **Painel utilizado para especificar agregações temporais (Painel 7) -** Este painel contém as classes temporais agregadas ao relacionamento que foi previamente selecionado pelo usuário, permitindo assim que o usuário elimine algumas classes desta lista, de forma a agrupar as instâncias de classes que não foram eliminadas;
8. **Painel 8 -** Painel que confirma ou cancela a condição temporal especificada pelo usuário.

A seguir descrevemos a seqüência de utilização dos operadores temporais presentes na janela *When?*

- Seqüência de utilização dos operadores temporais

1. O usuário seleciona um dos operadores temporais presentes no painel de condições temporais (painel 2);
2. Escolhe um dos operadores temporais: *All History*, *Instant* ou *Period*.

   2.1. *All History* – Ativa o painel de filtragens (painel 6) e o usuário se desejar, seleciona um dos operadores de filtragens: *First Interval, Specified Interval* ou *Last Interval*;

   2.2. *Instant* - Ativa o painel de granularidades temporais (painel 3) e o de especificar um instante de tempo (painel 4). Logo, o usuário realizará as seguintes tarefas:

   2.2.1. No painel 3, seleciona a granularidade desejada e especifica o instante de tempo baseado nesta granularidade;

   2.2.2. No painel 4, seleciona um dos operadores de instantes (*begin*, *at* e *end*) a ser atribuído a condição da consulta.

   2.3. *Period* - Ativa o painel de granularidades temporais (painel 3) e o de especificar um período de tempo (painel 5). Logo, o usuário realizará as seguintes tarefas:

   2.3.1. No painel 3, seleciona a granularidade desejada e especifica o início do período de tempo;

   2.3.2. No painel 5, seleciona a opção *from* para determinar o valor especificado no *passo 2.3.1* como início do período;

   2.3.3. No painel 3, seleciona a granularidade desejada e especifica o fim do período de tempo;

2.3.4. No painel 5, seleciona a opção *to* para determinar o valor especificado no *passo 2.3.3* como fim do período;

2.3.5. No painel 5, seleciona um dos operadores de período: *before(I,J), meet(I,J), cross(I,J), start(I,J), during(I,J), finish(I,J), cross(J,I), follow(I,J), after(I,J), contain(I,J), started-by(I,J)* e *finished-by(I,J)* a ser atribuído a condição da consulta.

3. Após especificar os operadores temporais o usuário clica no botão *Ok* ou *Cancel* (painel 8) para confirmação ou cancelamento da condição, respectivamente.

Quando o ícone *What?* é acionado, isto significa que o usuário deseja especificar alguma condição não temporal. A figura 3.7 ilustra o *layout* da janela *What?* quando o atributo nome *(name)* é selecionado. Esta janela é composta por um conjunto de painéis descritos a seguir.

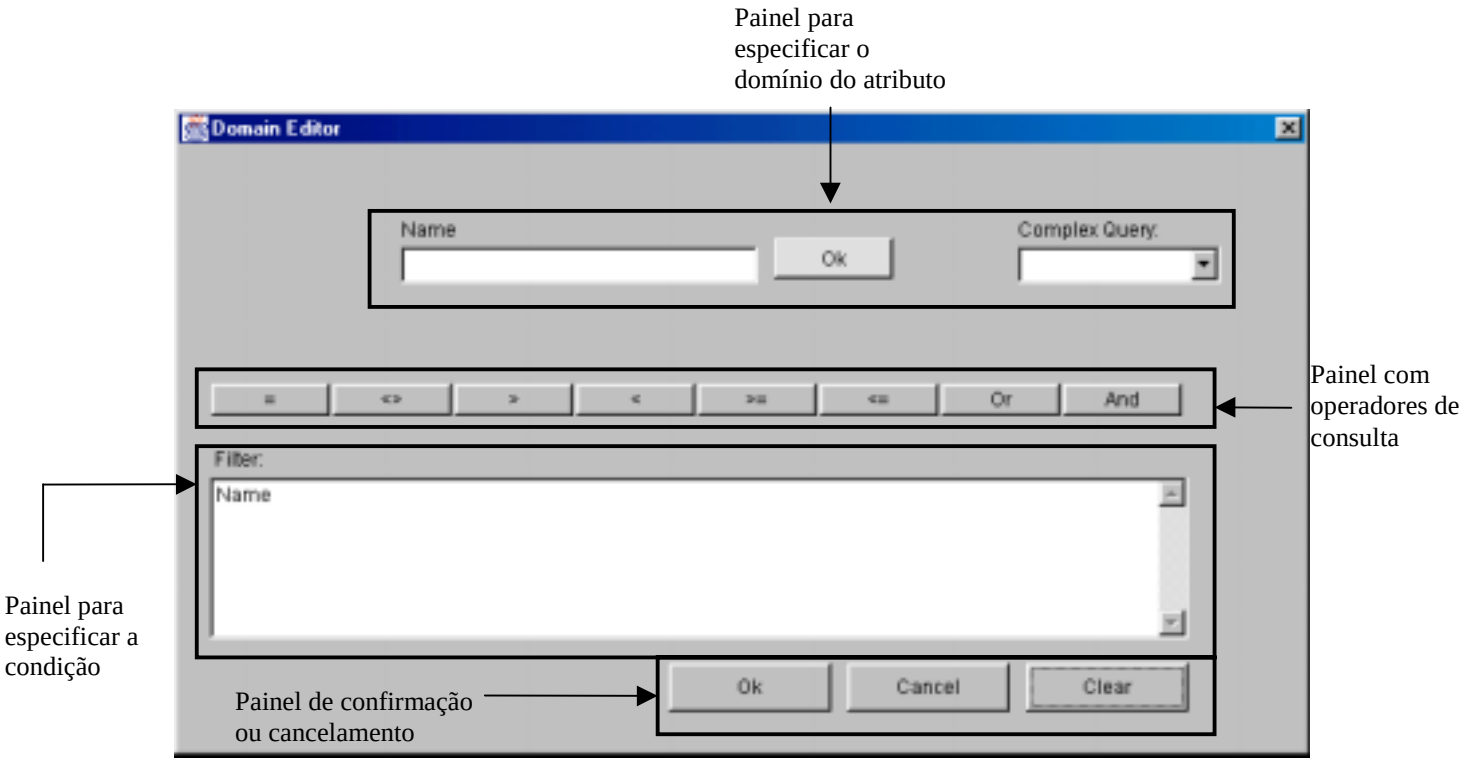


Figura 3.7 – *Layout* do painel *What?*

1. **Painel para especificar o domínio do atributo –** Este painel é composto por outros dois painéis. O primeiro painel é formado por uma faixa/lista de valores a serem escolhidos pelo usuário ou uma área para especificação do valor, dependendo do domínio (caracter ou número) do atributo selecionado. O segundo painel contém uma lista de atributos selecionados que fazem parte das consultas previamente

definidas pelo usuário e que irão compor a condição de uma consulta complexa, a seção 3.5.2 explicará em maiores detalhes as consultas complexas;

2. **Painel com operadores de consulta –** Este painel contém um conjunto de operadores a serem escolhidos pelo usuário. Os operadores dividem-se em duas categorias, os lógicos: AND e OR, e os operadores de comparação: = (igual), <> (diferente), < (menor), > (maior), <= (menor igual) e >= (maior igual);
3. **Painel para especificar a condição da consulta –** Este painel contém um área destinada a especificação (construção) da condição da consulta pelo usuário;
4. **Painel de Confirmação ou Cancelamento –** Este painel contém os botões para: cancelamento da condição *(Cancel)*, confirmação da condição *(Ok)* e reconstrução da condição especificada *(Clear)*.

As figuras 3.8 (a), 3.8 (b) e 3.8 (c) ilustram a utilização da janela *What?* para especificar a condição da seguinte consulta no TVQE: *Qual o salário do empregado Francisco de Assis?* Note que a condição desta consulta é *nome do empregado igual a Francisco de Assis*, ou seja, *Name = "Francisco de Assis"*. Logo, para construir esta condição no TVQE, selecionamos o atributo *Name* e acionamos o ícone *What* para visualização da janela *What?*. Nesta janela, fazemos o seguinte:

1. No painel de operadores de consulta, selecionamos o operador de igualdade, veja figura 3.8 (a);
2. No painel para especificar o domínio do atributo, especificamos o valor a ser atribuído ao atributo *Name*, ou seja, *Francisco de Assis*, em seguida selecionamos o botão de *Ok* para confirmar este valor, veja figura 3.8 (b);
3. No painel para especificar a condição, visualizamos a condição desta consulta, ou seja, *Name = Francisco de Assis*, veja figura 3.8 (c);
4. No painel de confirmação ou cancelamento, podemos: selecionar o botão *Ok*, *Cancel* ou *Clear*, para confirmar, cancelar ou reconstruir a condição desta consulta, respectivamente. Neste caso, selecionamos o botão *O*k para confirmação.

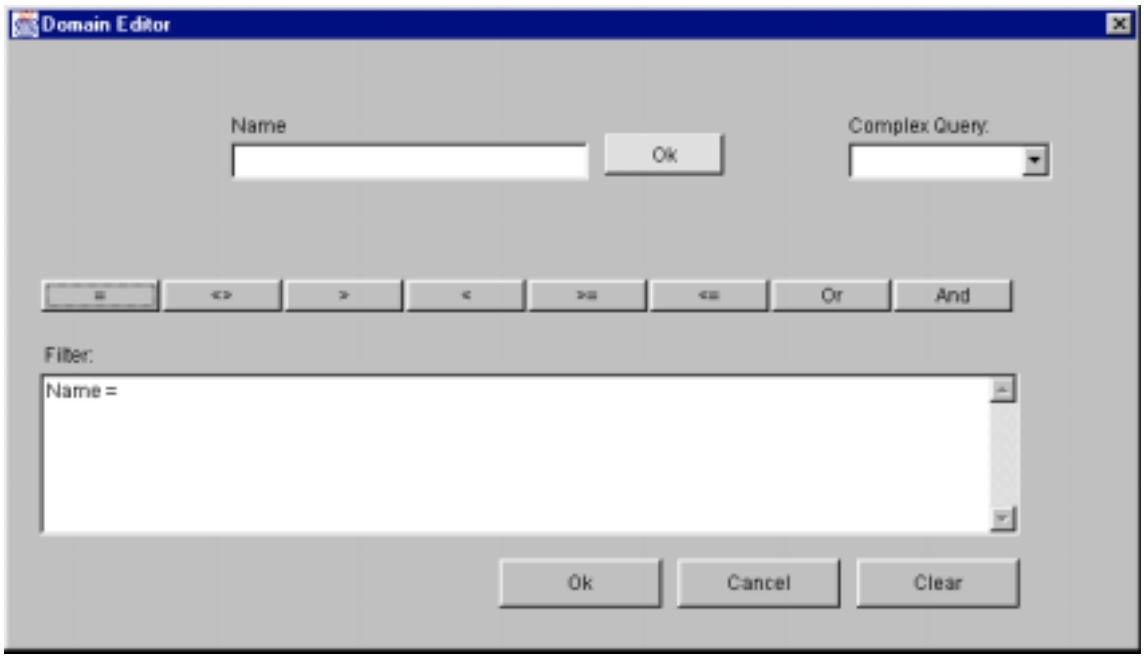


Figura 3.8 (a) – Especificação do operador de igualdade

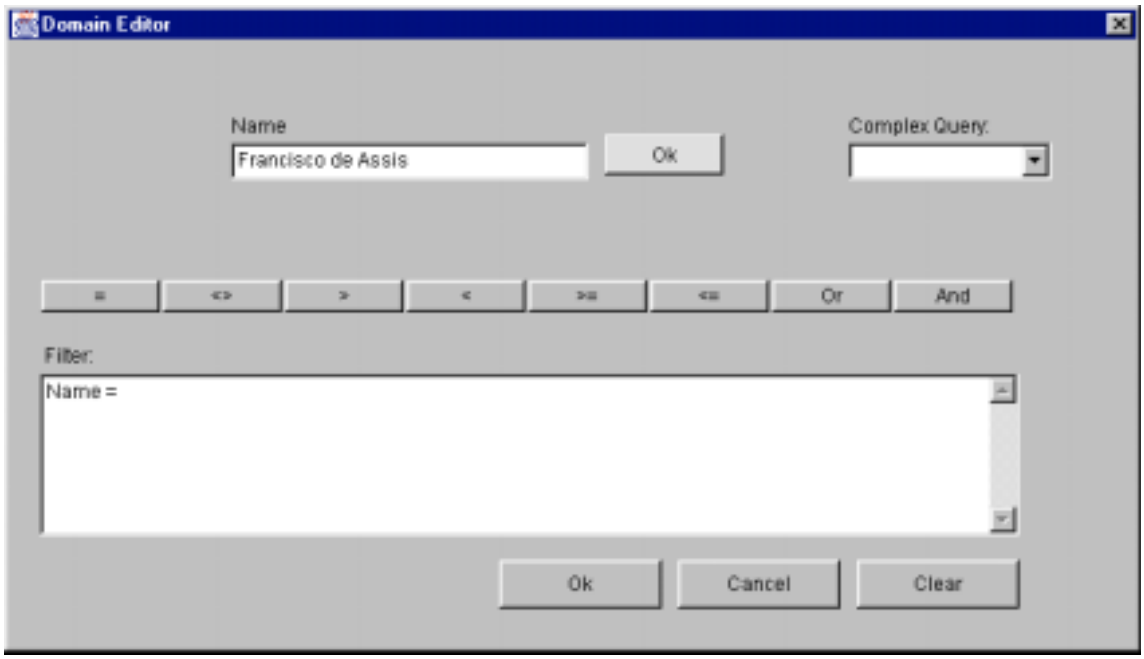


Figura 3.8 (b) – Especificação do valor, *Francisco de Assis*, para o atributo *Name*

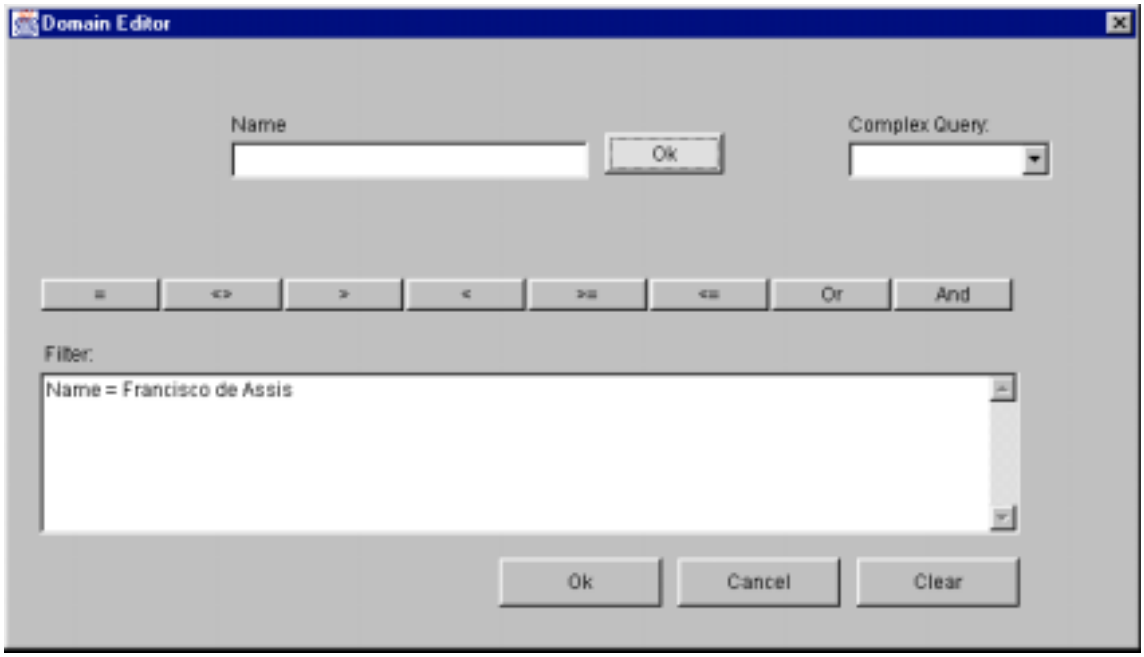


Figura 3.8 (c) – Visualização da condição *Name = Francisco de Assis*

A **saída** da informação corresponde ao resultado da consulta realizada pelo usuário. Este resultado pode ser visualizado através de uma representação intensional e extensional dos dados. A representação intensional consiste de um esquema da consulta (*query schema*), situado na janela de esquemas. A representação extensional é visualizada através de uma forma tabular, ou seja, os dados são mostrados em uma tabela.

Com a apresentação do ambiente TVQE podemos agora relacionar as primitivas gráficas temporais e suas respectivas operações na interface TVQE. A tabela 3.1 mostra as primitivas gráficas e seus correspondentes mecanismos visuais usados na interface TVQE [24]. Note que a abreviação *ET* significa extensão temporal.

Primitivas Gráficas	Interface Tvqe
Seleção de um nodo	Seleção sucessiva sobre o índice representando o nodo
Desenho de um arco entre os nodos *n* e *q*	Seleção sobre os índices representando *n* e *q* e o ícone *What?*
Mudança do rótulo de um arco entre os nodos *s* e *q*	Seleção sobre o índice representando *q* e o ícone *What?*
Seleção de nodo(s) sombreado(s)	Seleção sobre os índices representando nodo(s) e o ícone *When?*
ET da mudança do rótulo de um arco entre o nodo *s* e o nodo-papel *q*	Seleção sobre o índice representando *q* e o ícone *When?* Seleção das opções *instant* ou *period* Especificação da condição temporal no painel *When?*
Mudança do rótulo de um nodo	Seleção sobre o índice representando o nodo e o ícone *When?* Seleção do *checkbox "filter the retrieved time intervals"* Seleção do intervalo de tempo correspondente
Seleção do rótulo de um nodo	Seleção sobre o índice representando o nodo e o ícone *When?* Seleção do *checkbox "Temporal aggregation"* Seleção de classes na lista para serem eliminadas
ET do desenho de um arco entre os nodos *n* e *q*	Seleção sobre o índice representando o nodo e o ícone *When?* Seleção do *checkbox "Temporal reference to..."* e da classe representando *q* no *menu* de referências temporais Seleção das opções *instant* ou *period* Especificação da condição temporal no painel *When?*

Tabela 3.1 – Primitivas gráficas temporais e as correspondentes ações dentro da interface TVQE

Agora que já conhecemos o ambiente TVQE, partimos para a apresentação das modificações e extensões realizadas no mesmo como parte da presente dissertação.

3.4. Modificações no TVQE

As modificações realizadas no TVQE foram a sincronização dos botões e ícones da janela principal e dos painéis do janela temporal *When?*, além da criação de métodos

de ajuda ao usuário, como a orientação no uso da interface. A seguir descreveremos cada uma dessas modificações.

3.4.1. Sincronização realizada no TVQE

As sincronizações realizadas no TVQE estão relacionadas aos botões e ícones da janela principal (figura 3.2) e aos painéis da janela de condições temporais *WHEN?* (figura 3.6).

- Sincronização dos Botões e Ícones da Janela Principal

A sincronização dos botões e ícones da janela principal é um dos aspectos necessários para evitar que o usuário realize algumas ações "indesejadas", ou seja, cometer erros que possam comprometer a funcionalidade do sistema.

A sincronização da janela principal do TVQE é feita da seguinte forma: quando o usuário ativa o botão *Open* e abre um esquema de banco de dados para consulta, então os botões *Save* e *Close* são ativados. O ícone *When?* só é ativado quando é selecionado na agenda "gráfica" um índice que represente uma classe, relacionamento ou atributo temporal e o *What?* só é ativado quando é selecionado um atributo temporal ou não. Os ícones *Temporal Vis* e *Snapshot Vis* são ativados quando os índices que representam classes ou atributos, temporais ou não, são selecionados ou estão no estado visualizado.

- Sincronização dos Painéis para Especificação de uma Consulta Temporal: Painel *When?*

Como já foi visto na seção 3.5.2 a janela *When?* é composta de vários painéis que são ativados de acordo com a condição temporal selecionada pelo usuário. A tabela 3.2 ilustra a sincronização desses painéis.

Condição Temporal	Ação
All History	Habilita o painel de filtragem
Instant	Habilita o painel de granularidades temporais, de filtragem e o painel para especificar um predicado sobre um instante de tempo. Além de habilitar a condição *Temporal Reference to...*
Period	Habilita o painel de granularidades temporais, de filtragem e o painel para especificar um predicado temporal sobre um período de tempo. Além de ativar a condição *Temporal Reference to...*

Tabela 3.2 – Sincronização dos painéis da janela *When?*

3.4.2. Orientação no Uso da Interface

Com o objetivo de ajudar o usuário a interagir melhor com o ambiente TVQE, foi criado um painel de mensagens, localizado na parte inferior da janela principal do TVQE. Este painel contém uma série de mensagens que servem como guia do usuário, orientando-o passo a passo como proceder após a realização de alguma tarefa. Como exemplo, temos uma mensagem que indica o próximo passo a ser tomado pelo usuário após acionar o botão *Open* da janela principal e abrir um esquema do BD. A mensagem é a seguinte: "*Click on the Notebook index to choose a trigger class"* (Clique no índice da agenda para escolher uma classe alvo). Uma outra maneira de orientar o usuário é através do ícone de ajuda que se encontra na área de ícones (figura 3.2), mas ainda não foi implementado. A tabela 3.3 ilustra algumas mensagens presentes no sistema que orientam o usuário na sua interação com o mesmo.

Mensagens de Orientação		
Objeto	*Ação do usuário sobre o objeto*	*Mensagem após ação*
Estado inicial do Ambiente TVQE	-	Clique no botão *Open* para abrir um esquema para consulta
Botão *Open*	Acionar o botão *open* e abrir um esquema de consulta	Clique no índice da agenda para escolher uma classe alvo
Botão *Close*	Acionar o botão *close*	Clique no botão *Open* para abrir um esquema para consulta
Painel do Esquema Gráfico	Mudar do painel que contém o esquema gráfico para o esquema de consulta	Clique no botão *Save* para salvar o esquema gráfico
Índice da agenda	Seleciona um índice da agenda que representa uma classe, relacionamento ou atributo temporal	Clique no botão *When?* para escolher uma condição temporal
	Selecionar um índice da agenda que representa um atributo temporal ou não	Clique no botão *What?* para escolher uma condição
	Selecionar um índice da agenda que representa uma classe ou contexto	Escolha um atributo para determinar sua condição se existir

Tabela 3.3 – Mensagens presentes no painel de mensagens

3.5. Extensões no TVQE

As extensões realizadas no TVQE foram a criação de um esquema de consulta (*query schema*), a geração de consultas complexas e a versão *applet* do TVQE. A seguir serão descritas cada uma dessas extensões.

3.5.1. Criação do Esquema de Consulta *(Query Schema)*

O esquema de consulta representa um subesquema de interesse que compreende apenas as classes, atributos e relacionamentos selecionados pelo usuário durante o processo de formulação da consulta, além das condições temporais e/ou não temporais que o usuário poderá informar. Este esquema contém, como extensão, a resposta à consulta formulada. A figura 3.9 ilustra um exemplo de um esquema de consulta da agência de empregos, onde a seguinte consulta foi realizada: "*Quais os nomes dos empregados maiores de 18 anos*". Note que o nodo *Age* (idade do empregado) é o alvo da nossa condição e apenas os nodos selecionados pelo usuário são destacados.

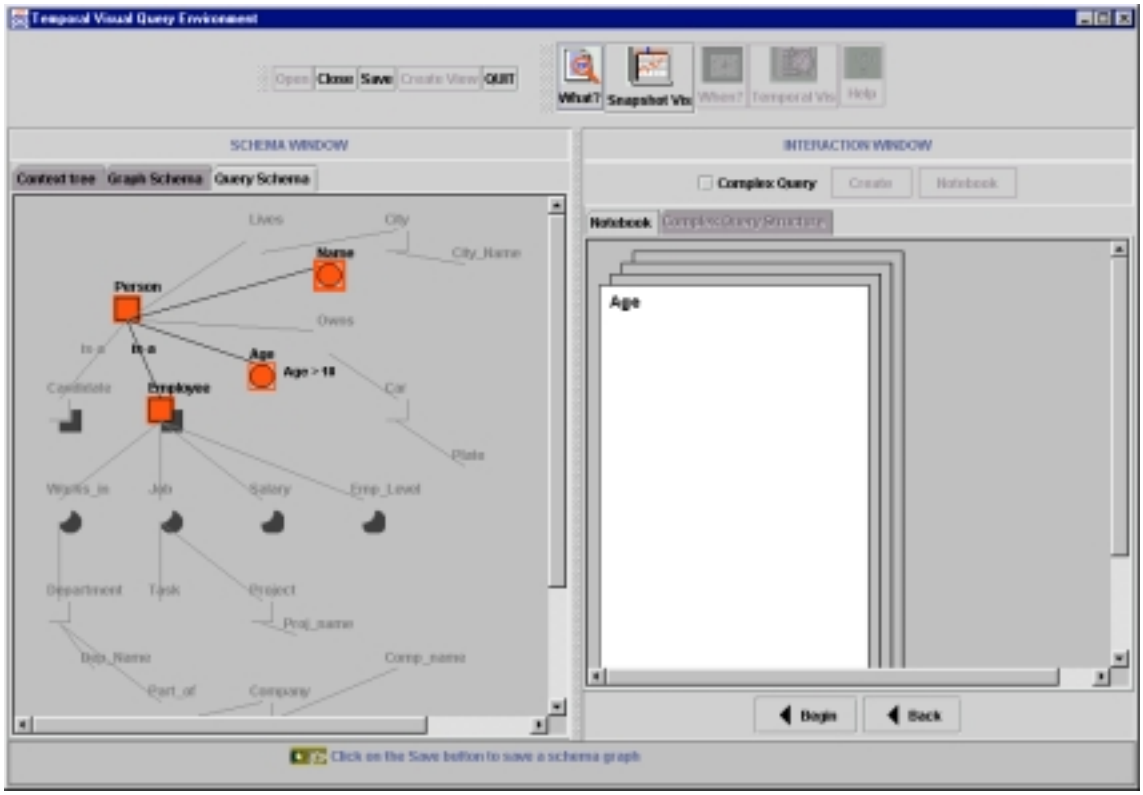


Figura 3.9 – Esquema de Consulta (*Query Schema*)

3.5.2. Criação de Consultas Complexas

Chamamos de consultas complexas, aquelas cujas condições se baseiam no resultado de outras consultas. No ambiente TVQE consultas complexas são criadas quando o *checkbox* de consultas complexas (*Complex Query)* é ativado e selecionado o botão *Create* (Criar). Esta ação permite criar consultas que podem ser referenciadas a posteriori como condições de outras consultas, isto é representado visualmente com a criação de um botão rotulado com o mesmo nome do último índice selecionado pelo usuário na agenda, este botão contém informações (esquema de consultas e seus respectivos índices na agenda) sobre esta consulta. Toda vez que o usuário quiser obter informações sobre uma consulta específica é só selecionar o botão desejado que se encontra no painel de consultas complexas (*Complex Query*), na janela de interação.

Para um melhor entendimento, vejamos um exemplo ainda com relação a agência de empregos descrita no início do capítulo. *Quais os nomes dos empregados cujos salários são maiores que o de João?* Analisando este exemplo, podemos dividir esta consulta em duas subconsultas: a **primeira** que recupera o salário de João, e a **segunda** que recupera

os nomes dos empregados cujos salários são iguais ao de João. Sabemos que a primeira consulta faz parte da condição da segunda, logo teremos que construí-la antes para depois formularmos a segunda. O processo de formulação dessas consultas no TVQE está descrito abaixo.

Na **primeira** parte da consulta temos que recuperar o salário de João, assim selecionamos as classes e atributos envolvidos, neste caso, temos: pessoa *(person)*, empregado *(employee)*, nome *(name)* e salário *(salary)*, acionamos o ícone *What?* e no editor de domínios atribuímos a condição *name = Joao*, ativamos *o checkbox* de consultas complexas *(Complex Query*) na janela de interação *(Interaction Window)* e selecionamos o botão de criação de consultas *(Create)*. A figura 3.10 (a) ilustra este processo.

Agora, vamos ao resultado final, ou seja, a formulação da **segunda** parte da consulta (*Quais os nomes dos empregados cujos salários são maiores que o de João*). Como já dispomos do resultado da primeira consulta, precisamos agora selecionar as classes e atributos envolvidos, neste caso, temos: *person*, *employee*, *name* e *salary*, e formulamos a condição: *salary>employee1.Salary*, onde *employee1.Salary* corresponde a referência ao atributo *Salary* da consulta anteriormente definida, denominada *employee1*. A figura 3.10 (b) ilustra a especificação da condição da consulta complexa e a figura 3.10 (c) ilustra o esquema final desta consulta.

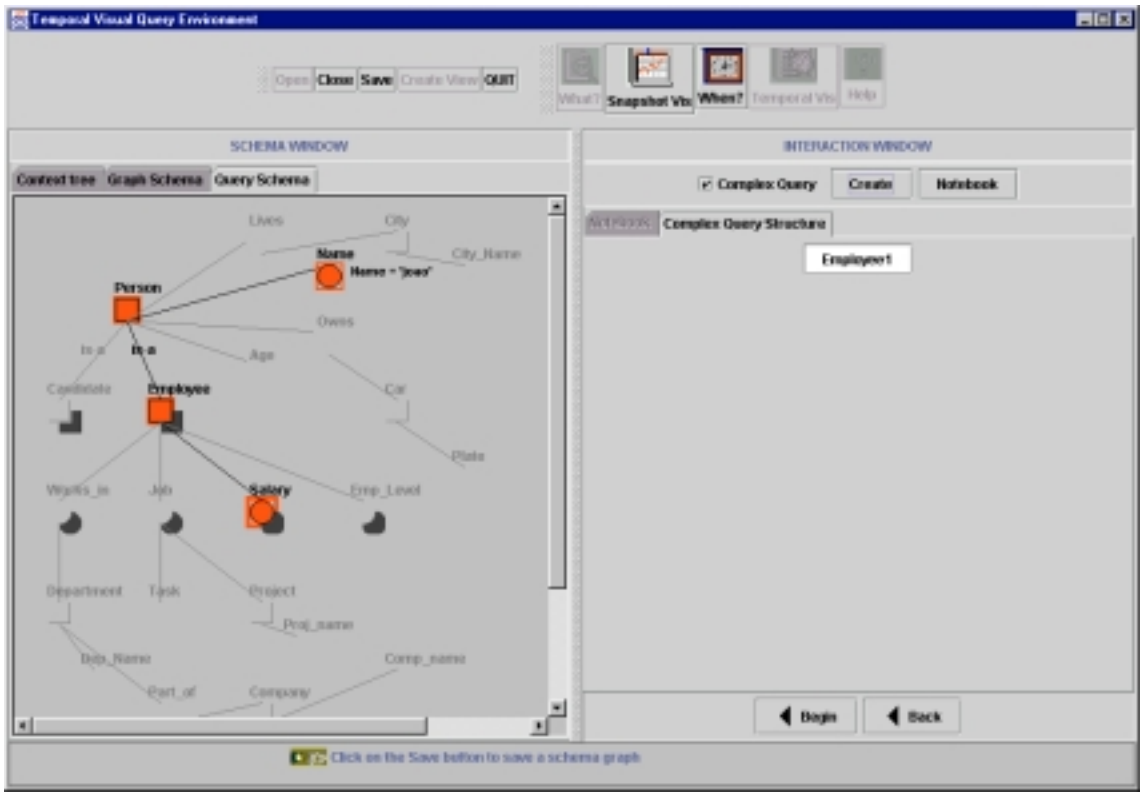


Figura 3.10 (a) – Especificação da consulta que faz parte da condição da consulta complexa

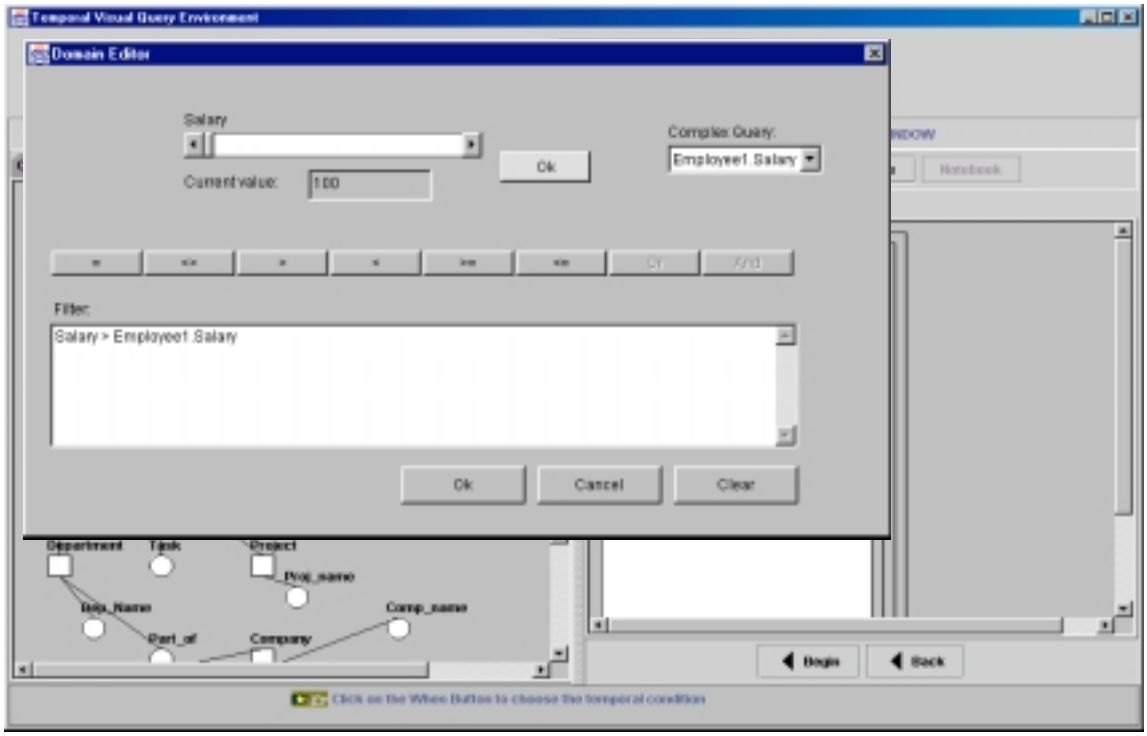


Figura 3.10 (b) – Especificação da condição da consulta complexa

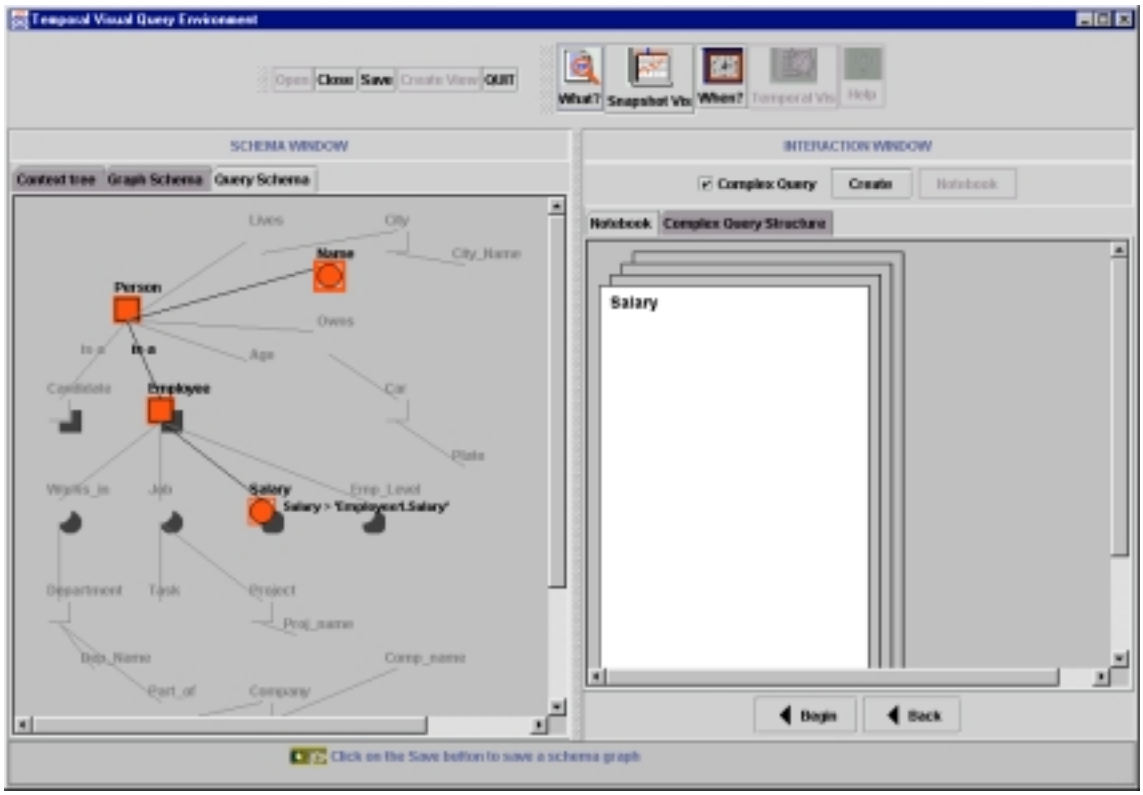


Figura 3.10 (c) – Esquema final da consulta complexa

3.5.3. O *Applet* TVQE

Com o objetivo de tornar a aplicação TVQE acessível a qualquer usuário que acessa a Internet, de modo a ser útil para um banco de dados público e também para receber críticas e sugestões sobre o ambiente TVQE, decidimos criar uma versão *applet*[6] do mesmo.

O versão *applet* TVQE não tem todas as funcionalidades da aplicação *stand-alone*. Primeiro, foram desativados os botões *Open, Close, Save, Create View e Quit*. Apenas o esquema da agência de empregos descrito no início deste capítulo foi usado como exemplo, não permitindo assim que outros esquemas sejam manipulados e, por último, o módulo Tradutor da aplicação *stand-alone*, a ser descrito no próximo capítulo, não foi implementado no *applet* TVQE.

[6] Informações sobre *applets* estão descritas no capítulo 5 deste trabalho.

Os componentes gráficos utilizados para construção da interface do *applet* TVQE não são os mesmos da aplicação *stand-alone* por questões de portabilidade, ou seja, o *applet* tem que executar em qualquer navegador *web*. A figura 3.11 ilustra o *layout* da tela principal do *applet* TVQE que está disponível para acesso no seguinte endereço: http://www.dsc.ufpb.br/~tvqe. As informações contidas no *layout* do *applet* TVQE, tais como as áreas de *menu*, ícones, títulos, visualização e interação, são as mesmas da versão *stand-alone*.

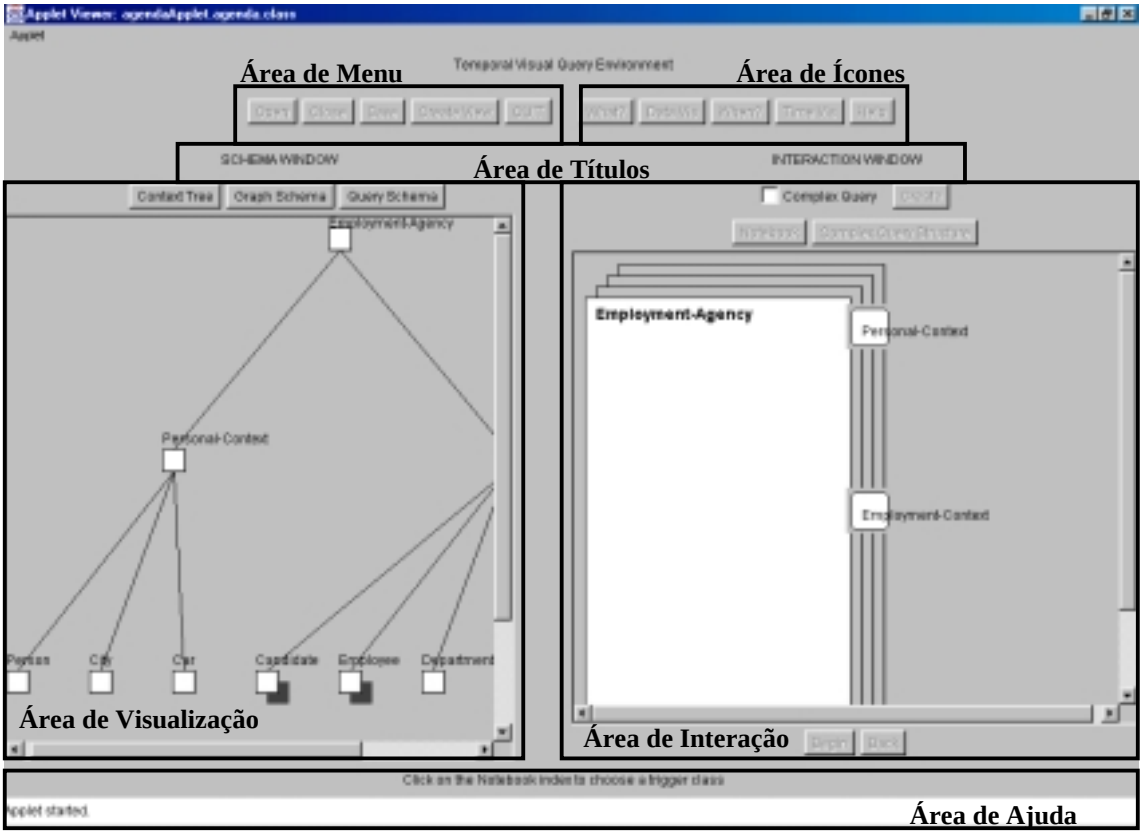


Figura 3.11 – *Layout* do *Applet* TVQE

3.6. Conclusão

Neste capítulo, apresentamos o ambiente TVQE, considerando sua funcionalidade, modificações e extensões realizadas. Através da explicação de cada *layout* da interface TVQE e de um exemplo ilustrativo de um esquema gráfico de uma agência de empregos.

As modificações feitas no TVQE tiveram o objetivo de melhorar a interação com o usuário, como por exemplo, mensagens de orientação de como proceder após a realização de alguma tarefa. Além da sincronização das ações do usuário através da ativação e desativação de algumas funções do sistema.

As extensões realizadas foram a criação do esquema de consulta (*query schema*), cuja finalidade é gerar para o usuário uma visualização intensional do resultado de sua consulta. As demais extensões foram a inclusão de consultas complexas e a versão *applet* do TVQE. As consultas complexas surgiram devido à necessidade do usuário formular consultas mais elaboradas cujas condições referenciam resultados pertencentes a outras consultas. A versão *applet* do TVQE ainda apresenta algumas restrições tanto relacionadas à interface quanto à funcionalidade, se comparado com a aplicação *stand-alone*. Apesar dessas restrições, sua funcionalidade não foi comprometida como um todo.

Capítulo 4 – Tradutor de Consultas Visuais

A maioria dos sistemas visuais de consultas dispõem de mecanismos visuais próprios de consultas, utilizando-se de diferentes representações (diagramáticas, icônicas, tabular, entre outras).

O TVQE é um sistema visual de consulta que utiliza essas representações visuais para que o usuário realize suas consultas sem a necessidade de conhecer uma linguagem textual. Mas para que esta consulta seja compreendida por um SGBD qualquer no qual está o banco de dados, é necessário um tradutor que transforme estas consultas para a linguagem do SGBD. Assim, surgiu o **Tradutor**, que realiza o mapeamento das consultas feitas no ambiente TVQE para uma linguagem textual de consulta, no caso SQL [14], pois é a linguagem padrão conhecida por qualquer SGBD. Como não existe nenhum software comercial específico para banco de dados temporais optou-se pela linguagem SQL, que garante a portabilidade para qualquer SGBD relacional. Neste caso, as informações temporais representadas no SGBD relacional devem estar providas dos atributos temporais adequados.

Este capítulo tem como principal objetivo a descrição do Tradutor. Considerando sua arquitetura, o mapeamento do esquema TGM (*Temporal Graph Model*) para o relacional e o mapeamento das consultas realizadas pelo usuário no TVQE para a linguagem SQL.

4.1. Arquitetura do Tradutor

O Tradutor interage com vários componentes da arquitetura do TVQE. A figura 4.1 ilustra esta arquitetura. A **Interface TVQE** forma o ambiente no qual o usuário irá interagir para formulação de sua consulta. Para isso, o usuário utiliza-se do **Esquema TGM** que serve como instrumento básico, de modo a guiar o usuário para a formulação da consulta, sendo gerado por uma ferramenta de projeto conceitual (CASE). O apêndice A ilustra a estrutura interna deste esquema. A medida que a consulta estiver sendo realizada, novos subesquemas do esquema TGM são gerados com apenas os nodos selecionados durante o processo de formulação da consulta até gerar o esquema de consulta (**Subesquema TGM**). A partir do esquema de consulta, o **Tradutor** transforma a consulta gerada para a linguagem SQL. Isto é possível porque o Tradutor recebe como entrada a **Estrutura TGM-Relação**, que relaciona cada componente do esquema gráfico TGM com seu correspondente no modelo relacional. Esta estrutura é criada a partir do módulo **Mapeamento TGM-Relação**. Assim o Tradutor lê o esquema de consulta e a estrutura TGM-Relação gerando a consulta na linguagem SQL, que é entregue a um SGBD através do **Dispositivo JDBC**[7] para então ser processada e seu resultado retornado ao usuário.

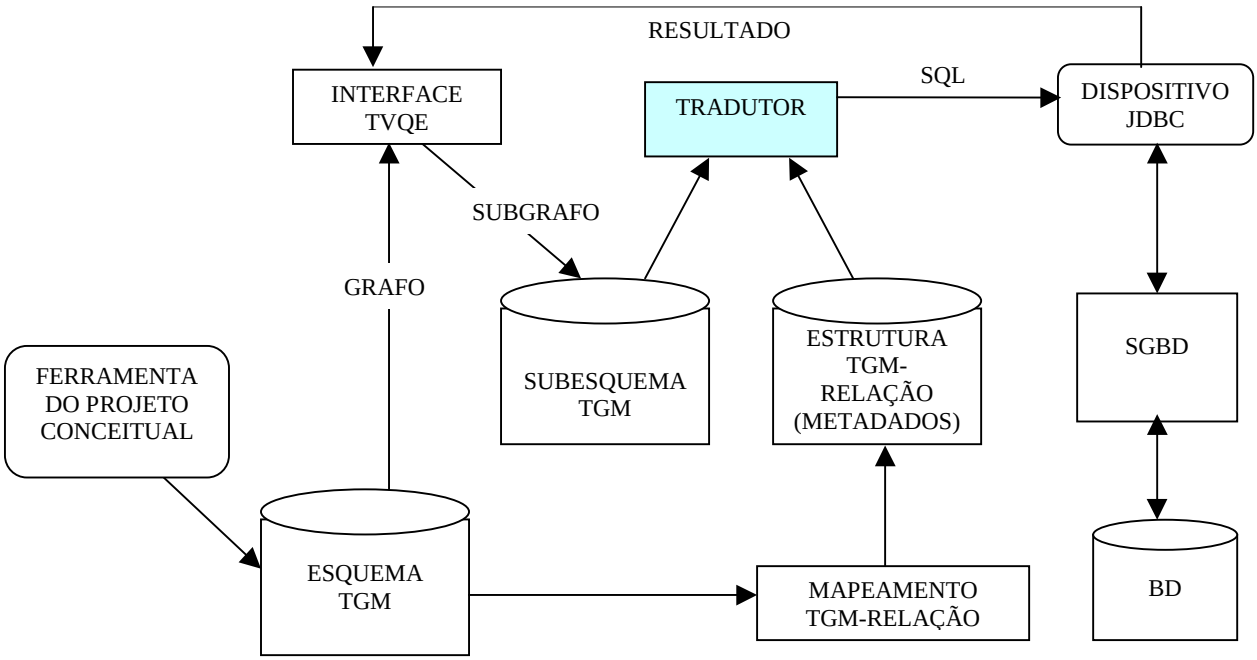


Figura 4.1 – Arquitetura funcional do TVQE

[7] É um dispositivo responsável por fazer a conexão com o banco de dados, ver capítulo 5.

O Tradutor necessita de uma versão relacional do TGM. Para tal foram definidas regras de mapeamento do esquema TGM para o relacional, pois a partir destas regras pode-se criar a estrutura TGM-Relação e o próprio banco de dados.

4.2. Mapeamento do Esquema TGM para Modelo Relacional

A implementação de um banco de dados temporal usando um SGBD relacional requer uma estratégia específica para a sua representação de modo a tornar o gerenciamento dos dados históricos independente da intervenção do usuário ou programador de aplicações [17]. Para tanto o SGBD deverá prover alguns recursos como, por exemplo, um sistema de regras de propósito geral ou um mecanismo de navegação sobre os dados históricos. Apesar de algumas limitações, é possível a utilização de SGBDs relacionais que não possuam tais facilidades, transferindo-se, neste caso, algumas tarefas para os programas de aplicação.

O mapeamento do esquema TGM para o modelo relacional, baseia-se no mapeamento do modelo de Entidade e Relacionamento (ER) para o relacional. Originalmente, o modelo ER [5] é formado basicamente por entidades, relacionamentos e atributos, mais tarde, nas versões estendidas do ER, foram incluídos outros conceitos, tais como as hierarquias de generalização/especialização, agregação e agrupamento.

Segundo [5], as **entidades** representam classes de objetos do mundo real, por exemplo, *pessoa, cidade, carro*. Os **relacionamentos** representam agregações de duas ou mais entidades, por exemplo, o relacionamento *Mora* relaciona as entidades, *pessoa* e *cidade*. Os **atributos** representam propriedades elementares de entidades ou relacionamentos, por exemplo, os atributos *Nome* e *Idade* da entidade pessoa.

Uma entidade E é uma **generalização** de um grupo de entidades E_1, E_2, ..., E_n, se cada E_1, E_2, ..., E_n é um subconjunto de E, por exemplo a entidade *pessoa* é um generalização das classes *empregado* e *candidato*. O inverso da generalização é a **especialização.** Uma entidade E é uma **agregação** de entidades E_1, E_2, ..., E_n, se cada E_1, E_2, ..., E_n for parte de E, ou seja, E_1, E_2, ..., E_n juntas formam E. Por exemplo, a entidade *data* é uma agregação de *dia, mês* e *ano*. No **agrupamento** consideramos conjuntos de entidades (do mesmo tipo) como entidades de grupo que podem ter seus próprios atributos ou relacionamentos, por exemplo, *uma classe escolar*, *um grupo de pesquisa, os filhos de um casal*, entre outros [31].

No esquema TGM, a **agregação** é representada através do relacionamento ***part-of*** (parte-de) e a **generalização/especialização** por uma seta rotulada com a palavra ***IS-A*** (é-um). O agrupamento não tem representação no esquema TGM, mesmo assim é considerado no mapeamento prevendo sua inclusão futura no TGM.

As cardinalidades do esquema TGM não são visualizadas no TVQE, mas fazem parte do conjunto de restrições de integridade ***c*** do TGMDB. Este conjunto é especificado pelo projetista de banco de dados, utilizando um linguagem apropriada, de forma a expressar condições e propriedades relevantes das classes e relacionamentos [24].

Note que os conceitos dos componentes que constituem o esquema TGM (esquema gráfico do TVQE), tais como: classes, relacionamentos e atributos, são equivalentes aos conceitos de entidades, relacionamentos e atributos, respectivamente, do modelo ER. É lógico que não incluímos nesta comparação os aspectos temporais do esquema TGM, pois não possuem equivalente no modelo ER. Logo, dividimos o mapeamento do esquema TGM para o relacional em duas partes: a primeira não considera os aspectos temporais e a segunda trata dos aspectos temporais do esquema.

O modelo relacional representa um banco de dados como uma coleção de relações. Informalmente, cada relação é uma tabela de tuplas e cada tupla representa uma coleção de dados relacionados. A seguir descreveremos o mapeamento do esquema TGM para o modelo relacional.

- Parte I do Mapeamento

Como já foi descrito acima esta parte do mapeamento não considera os aspectos temporais do esquema TGM. Baseado nos mapeamentos definidos em [6][30] temos as seguintes regras:

1. Para cada classe não-temporal, é criada uma relação com o mesmo nome;
2. Cada atributo não temporal é incluído como atributo na relação gerada a partir da classe no qual se relaciona;
3. Seja R um relacionamento N-1 (1-N), não-temporal, entre duas classes C_1 e C_2, então a chave primária da relação correspondente à classe cuja

cardinalidade é 1 é incluída na relação que representa a outra classe cuja cardinalidade é n;

4. Seja R um relacionamento 1-1, não temporal, entre as classes C_1 e C_2, então a chave primária da relação que representa a classe C_1 é incluída na relação da classe C_2 como atributo e vice-versa. Se, no modelo conceitual, um dos sentidos tiver cardinalidade mínima igual a zero, escolha a relação que corresponde à classe cujo relacionamento é zero para eliminar o atributo;
5. Para cada relacionamento N-M (ou seja, em que ambas as cardinalidades máximas sejam > 1), não-temporal, é criada uma relação especial cujo nome será o nome do relacionamento e dois atributos, cada um com o nome de uma das classes relacionadas que formarão a chave primária da relação. Se uma das classes relacionadas possui uma chave composta, os componentes da chave deverão aparecer como atributos distintos no lugar do nome da classe;
6. Para cada generalização é acrescentada a chave da superclasse como atributo nas subclasses. Note que, exceto a chave da superclasse, só devem aparecer como atributos os relacionamentos específicos da subclasse;
7. Para cada agregação acrescente as chaves das classes componentes como atributos da relação da classe agregada, pois deverão formar uma chave composta;
8. Para cada agrupamento disjunto acrescente a chave dos grupos como atributo na classe de elementos;
9. Para cada agrupamento não disjunto, cria uma relação especial que relaciona os elementos aos grupos. Terá como atributos as chaves da classe grupo e da classe elementos.

Os identificadores das classes não são visualizados no esquema TGM, mas tornam-se chaves primárias das relações geradas a partir destas classes no modelo relacional.

- Parte II do Mapeamento

Esta é a segunda parte do mapeamento do esquema TGM para o relacional, pois aqui vamos considerar os aspectos temporais. O termo intervalo utilizado a partir deste ponto da dissertação eqüivale à definição de período.

No TGM as classes e relacionamentos podem ser modelados como componentes temporais de banco de dados temporais [23], ou seja, o usuário recupera não somente os dados correntes, mas também os dados passados (históricos). Então, nesse contexto, a cada instância do banco de dados está associado um conjunto de intervalos de valores disjuntos, denominado **tempo de vida** (*lifespan*), cujo domínio de tempo é linearmente ordenado e discreto. Cada intervalo é representado por um par ordenado de números naturais. Em particular, assumimos que todo *lifespan* está normalizado, ou seja, seus elementos são disjuntos dois a dois.

Com o objetivo de eliminar a redundância dos dados em uma relação de tempo-válido, aplicamos a normalização temporal descrita em [19] nestas relações, onde cada atributo, cujo valor varie com o tempo, é decomposto em uma nova relação de tempo-válido. Note que esses atributos representam exatamente os atributos temporais presentes no esquema TGM.

Para realizar o mapeamento dos aspectos temporais no esquema TGM para o modelo relacional, temos as seguintes regras:

1. Para cada classe temporal, é criada uma relação com o mesmo nome;
2. Para cada atributo temporal ***A*** de uma classe ***C*** é criada uma relação, tendo como atributos os atributos chaves da relação criada a partir de ***C*** e o próprio atributo ***A***;
3. Para todo relacionamento temporal, é criada uma relação cujo nome será o nome do relacionamento e dois atributos, cada um com o nome de uma das classes relacionadas que formarão a chave primária da relação. Se uma das classes relacionadas possui uma chave composta, os componentes da chave deverão aparecer como atributos distintos no lugar do nome da classe;
4. Em cada relação criada (em 1, 2 e 3) são incluídos dois atributos temporais **de** e **ate**, que irão compor a chave primária da relação. Estes atributos permitem representar o tempo de vida dos objetos.

A granularidade mais fina dos atributos temporais no banco de dados, no qual o Tradutor irá acessar, é a hora. Isto foi definido devido a esta ser a mesma considerada pelo ambiente TVQE nas condições temporais das consultas. No entanto, para cada tabela, a granularidade pode ser especificada de forma variada, ou seja, AAAA, AAAAMM, AAAAMMDD e AAAAMMDDHH, dependendo do formato dado aos atributos temporais de cada tabela temporal. Onde AAAA, representa o ano, MM o mês, DD o dia e HH a hora.

Para representar o valor temporal ***Agora*** nos atributos temporais (**de** e **ate**) das relações no banco de dados dispomos de duas soluções. Na primeira, representamos o valor ***Agora*** através do valor numérico ***9999999999***, indicando AAAAMMDDHH, já que os atributos temporais são constantemente comparados com valores numéricos. Na segunda solução incluímos uma função no banco de dados que transforma o valor ***Agora*** na data atual do sistema, sempre que encontrá-lo. A solução adotada fica a critério do projetista do banco de dados. Note que na primeira solução apenas é possível processar dados passados e presentes, já que o valor numérico ***9999999999*** representa o momento presente. Enquanto na segunda solução os dados passados, presentes e futuros podem ser considerados, pois o valor ***Agora*** sempre é convertido para a data atual do sistema. Por motivos de simplicidade, adotamos a primeira solução.

A Estrutura TGM-Relação originada a partir do mapeamento do esquema TGM para o modelo relacional tem o formato ilustrado na tabela 4.1. Esta estrutura relaciona cada componente do esquema TGM com um do modelo relacional (relações, atributos, chaves primárias, chaves estrangeiras, etc). A tabela 4.2 ilustra a notação utilizada para indicar a representação de um componente do esquema TGM com o seu correspondente no modelo relacional.

Formato da Estrutura TGM-Relação
Nome da classe no esquema TGM + **0** + Nome da relação
Nome da classe, relacionamento ou atributo temporal no esquema TGM + **1** + Nome da relação de tempo-válido + Atributos temporais
Nome do relacionamento (N:M) no esquema TGM + **0** + Nome da relação
Nome do atributo no esquema TGM + **2** + Nome da Relação
Nome do atributo temporal no esquema TGM + **2** + Nome da Relação de tempo-válido
Nome da classe ou relacionamento (temporal ou não) ou do atributo temporal no esquema TGM + **3** + Nome da chave estrangeira + Nome da relação na qual é chave estrangeira + Nome da relação na qual a chave origina + Nome da chave na relação original

Tabela 4.1 – Formato da Estrutura TGM-Relação

	Componente no Modelo Relacional
0	Relação
1	Relação de Tempo-Válido
2	Atributo
3	Chave Estrangeira

Tabela 4.2 – Notação

Como exemplo deste mapeamento podemos utilizar o esquema gráfico TGM da agência de empregos, figura 4.2, cuja estrutura TGM-Relação está descrita na tabela 4.3.

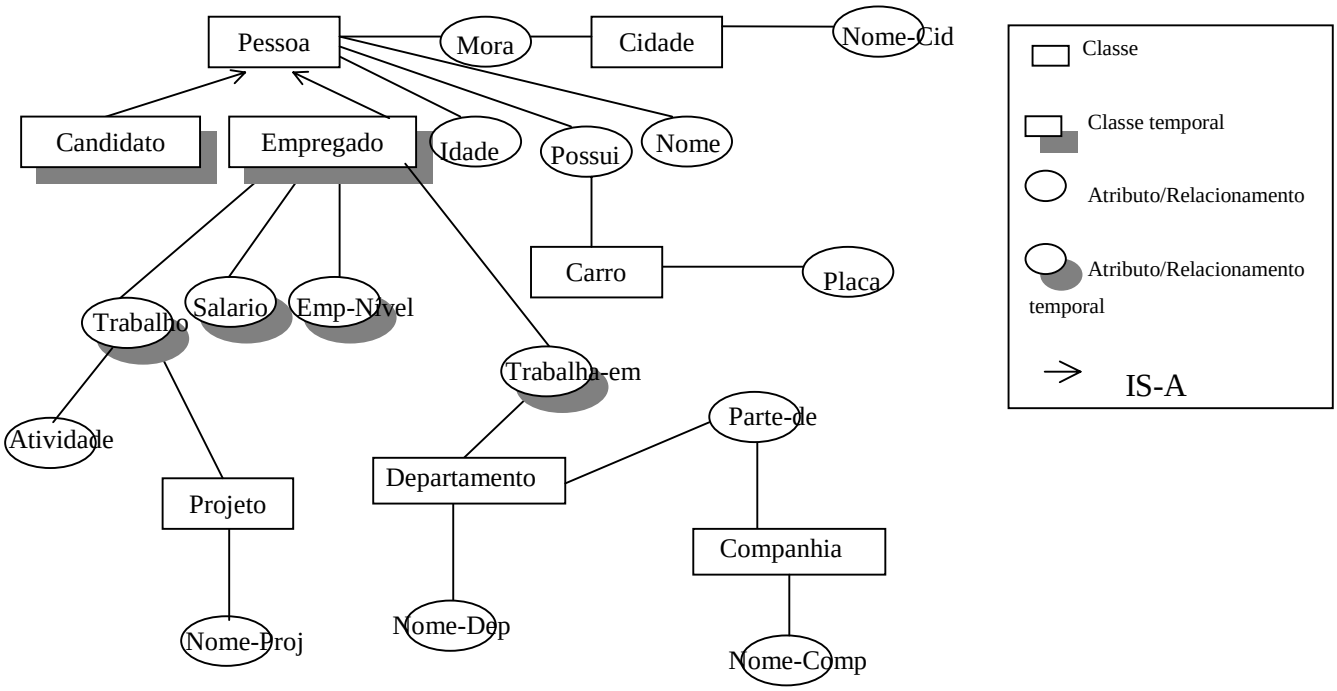


Figura 4.2 – Esquema gráfico TGM de uma agência de empregos

A seguir descrevemos as cardinalidades dos relacionamentos entre classes do esquema TGM da figura 4.2, pois fazem parte do conjunto de restrições de integridade *c* do TGMDB.

Relacionamento **Mora** {
Pessoa → Cidade (1:1)
Cidade → Pessoa (1:N)
}

Relacionamento **Possui** {
Pessoa → Carro (1:N)
Carro → Pessoa (1:1)
}

Relacionamento **Trabalha-em** {
Empregado → Departamento (1:1)
Departamento → Empregado (1:N)
}

Relacionamento **Trabalho** {
Empregado → Projeto (1:N)
Projeto → Empregado (1:N)
}

Estrutura TGM-Relação
Pessoa 0 Pessoa
Empregado 1 Empregado de ate
Empregado 3 Emp_Id Empregado Pessoa PessoaId
Candidato 1 Candidato de ate
Candidato 3 CandidatoId Candidato Pessoa PessoaId
Mora 3 CidadeId Pessoa Cidade CidadeId
Cidade 0 Cidade
Nome-Cid 2 Cidade
Nome 2 Pessoa
Idade 2 Pessoa
Emp-Nivel 1 Empregado_Nivel de ate
Emp-Nivel 2 Empregado_Nivel
Emp-Nivel 3 Emp_Id Empregado_Nivel Empregado Emp_Id
Salario 1 Empregado_Salario de ate
Salario 2 Empregado_Salario
Salario 3 Emp_Id Empregado_Salario Empregado Emp_Id
Trabalha-em 1 Trabalha_Em de ate
Trabalha-em 3 DepartamentoId Trabalha_Em Departamento DepartamentoId
Trabalha-em 3 Emp_Id Trabalha_Em Empregado Emp_Id
Trabalha 1 Trabalha de ate
Trabalha 3 ProjectId Trabalha Project ProjectId
Trabalha 3 Emp_Id Trabalha Empregado Emp_Id
Project 0 Project
Nome-Proj 2 Projeto
Departamento 0 Departamento
Nome-Dep 2 Departamento
Companhia 0 Companhia

Nome-Comp 2 Companhia
Parte-de 3 CompId Companhia Departamento DepartamentoId
Carro 0 Carro
Possui 3 PessoaId Carro Pessoa PessoaId
Placa 2 Carro
Tarefa 2 Trabalha

Tabela 4.3 – Estrutura TGM-Relação para uma agência de empregos

Logo, de acordo com o mapeamento do esquema TGM para o relacional, temos o seguinte esquema do banco de dados relacional, descrito abaixo, onde *PessoaId*, *CandidatoId*, *Emp_Id*, *ProjetoId*, *CompId*, *CidadeId*, *DepId* e *CarroId* correspondem aos identificadores das classes *Pessoa*, *Candidato*, *Empregado*, *Projeto*, *Companhia*, *Cidade*, *Departamento* e *Carro*, respectivamente no esquema de uma agência de empregos (figura 4.2).

Pessoa (PessoaId, Nome, Idade, *CidadeId)
Candidato (*CandidatoId, de, ate)
Empregado (*Emp_Id, de, ate)
Projeto (*ProjetoId, Nome_Proj)
Trabalha (*Emp_Id, *ProjetoId, Atividade, de, ate)
Empregado_Salario (*Emp_Id, Salario, de, ate)
Empregado_Nivel (*Emp_Id, Emp_Nivel, de, ate)
Carro (CarroId, Placa, *PessoaId)
Cidade (CidadeId, Nome_Cid)
Trabalha_Em (*Emp_Id, *DepartamentoId, de, ate)
Departamento (DepartamentoId, Nome_Dep)
Companhia (*CompId, Nome_Comp)

Notação: O asterisco (*) representa uma chave estrangeira e o sublinhado representa chave primária.

Como já foi realizado todo o processo de mapeamento do esquema TGM para um esquema relacional, então podemos partir para o processo de especificação e mapeamento das consultas a ser descrito na próxima seção deste capítulo.

4.3. Mapeamento das Consultas no TVQE

O ambiente TVQE permite que o usuário realize tanto consultas convencionais como temporais. Estas consultas são transformadas em instruções SQL, de modo a ser processada por um SGBD. A seguir descrevemos estes tipos de consultas, restringindo as consultas temporais de acordo com uma taxonomia definida.

4.3.1. Uma Taxonomia para Consultas Temporais

Segundo Fernandes Silva [24], uma taxonomia para consultas temporais foi proposta em Jensen et al [4], onde uma consulta temporal tem dois componentes ortogonais: seleção temporal e projeção temporal. A **seleção temporal** é uma condição lógica, baseada em um predicado que envolve o tempo associado com os fatos. A **projeção temporal** retorna os valores de tempo associados aos dados derivados da seleção temporal.

Segundo Fernandes Silva [24], uma taxonomia mais detalhada foi proposta em Nina et al [16], onde as possíveis combinações entre seleção/projeção temporal sobre o tempo e dados foram analisados, resultando em: **seleção/projeção dos dados**, onde as condições e resultados se aplicam somente a valores de dados; **seleção/projeção temporal**, onde as condições e resultados se aplicam a valores temporais; e **seleção/projeção mista**, onde as condições e resultados se aplicam tanto aos dados quanto aos valores temporais. Esta análise foi combinada com cinco tipos de situações históricas identificadas em um banco de dados bitemporal.

Em um banco de dados bitemporal, tanto o tempo de transação quanto o tempo válido são armazenados, sobre os quais cinco situações históricas podem ser identificadas:

1. **Dados instantâneos atuais**, representados por todas as informações válidas no momento presente;
2. **Dados instantâneos passados**, representados pelos dados válidos em um determinado instante do passado, de acordo com a atual percepção da história do banco de dados;

3. **Dados instantâneos de história passada**, considerando todas as informações de um determinado momento no passado, de acordo com a história válida naquele momento;
4. **Dados históricos**, nos quais estão incluídas todas as informações do passado armazenadas de acordo com a presente percepção de dados válidos;
5. **Dados históricos de história passada**, análogos aos anteriores mas considerando uma percepção anterior à atual, definida por um determinado tempo de transação.

Dentro do contexto de banco de dados de tempo-válido, o que é o caso do ambiente TVQE, não são suportadas consultas dos tipos 3 e 5 da lista anterior. A tabela 4.4 ilustra os tipos de consultas temporais que o TVQE suporta, exemplificando várias delas.

	Dados Instantâneos Passados	**Dados Históricos**
Seleção dados/ projeção temporal	Ex.3	
Seleção dados/projeção mista		Ex.4
Seleção temporal/projeção dados	Ex.1	
Seleção temporal/projeção mista		Ex.6
Seleção mista/projeção dados	Ex.2	
Seleção mista/projeção temporal		Ex.7
Seleção mista/projeção mista		Ex.5

Tabela 4.4 – Consultas temporais suportadas pelo TVQE

Ex.1 – Quais foram os salários dos empregados em 10/01/98?

Ex.2 – Qual foi o salário de João quando ele mudou o seu *status* pela última vez?

Ex.3 – Desde quando o grupo de banco de dados trabalha no projeto “Interface para banco de dados temporais”?

Ex.4 – Qual o histórico salarial dos empregados?

Ex.5 – Qual o último projeto dos empregados que tiveram um salário inicial maior que 5.000 reais? E desde quando?

Ex.6 – Qual o histórico dos níveis dos empregados antes de 2000?

Ex.7 – Qual o período que João trabalhou no projeto “Interfaces Visuais Avançadas” durante 1998?

Dividimos as consultas realizadas no TVQE em duas categorias: as convencionais e as temporais. A seguir descreveremos cada uma dessas consultas.

4.3.2. Consultas Convencionais

As consultas convencionais são aquelas em que o usuário não especifica nenhum tipo de condição temporal para seus atributos, permitindo assim apenas a recuperação de dados correntes (atuais) do BD. Este tipo de consulta é expressa visualmente no TVQE quando o usuário formula sua consulta e ativa o ícone *Snapshot Vis*, para visualização dos dados.

Para um melhor entendimento vamos formular a seguinte consulta: *Quais os nomes e salários de todos os empregados que foram alocados para trabalhar no projeto "Interface visual para banco de dados temporal"?*, relacionada ao esquema TGM de uma agência de empregos.

Para construirmos esta consulta no TVQE fazemos o seguinte: selecionamos na agenda "gráfica" os índices contexto-pessoal *(personal-context)*, pessoa *(person)*, empregado *(employee)*, Nome do empregado *(name)*, salário *(salary)*, trabalho *(job)*, projeto *(project)* e nome do projeto *(proj_name)* e acionamos o ícone *What?* para especificar a condição *proj_name* = "Interface visual para banco de dados temporal" no editor de domínios, a figura 4.3 ilustra esta condição. Como queremos recuperar os salários e os nomes dos empregados, então os índices da agenda que representam salário *(salary)* e nome *(name)* ficam no estado visualizado (pintados e contornados de vermelho), pois irão fazer parte da cláusula SELECT, quando transformados para SQL.

Note que os índices *salary*, *job* e *employee* representam um atributo, um relacionamento e uma classe temporal, respectivamente, mas como a consulta é convencional apenas os dados correntes serão recuperados, ou seja, os dados cujos atributos temporais estiverem dentro do intervalo de tempo corrente (atual). A figura 4.4 ilustra o resultado intensional da consulta através de um esquema gráfico *(query schema)* e a figura 4.5 o resultado extensional da consulta, ou seja, os dados, obtido pelo acionamento do ícone *Snapshot Vis*.

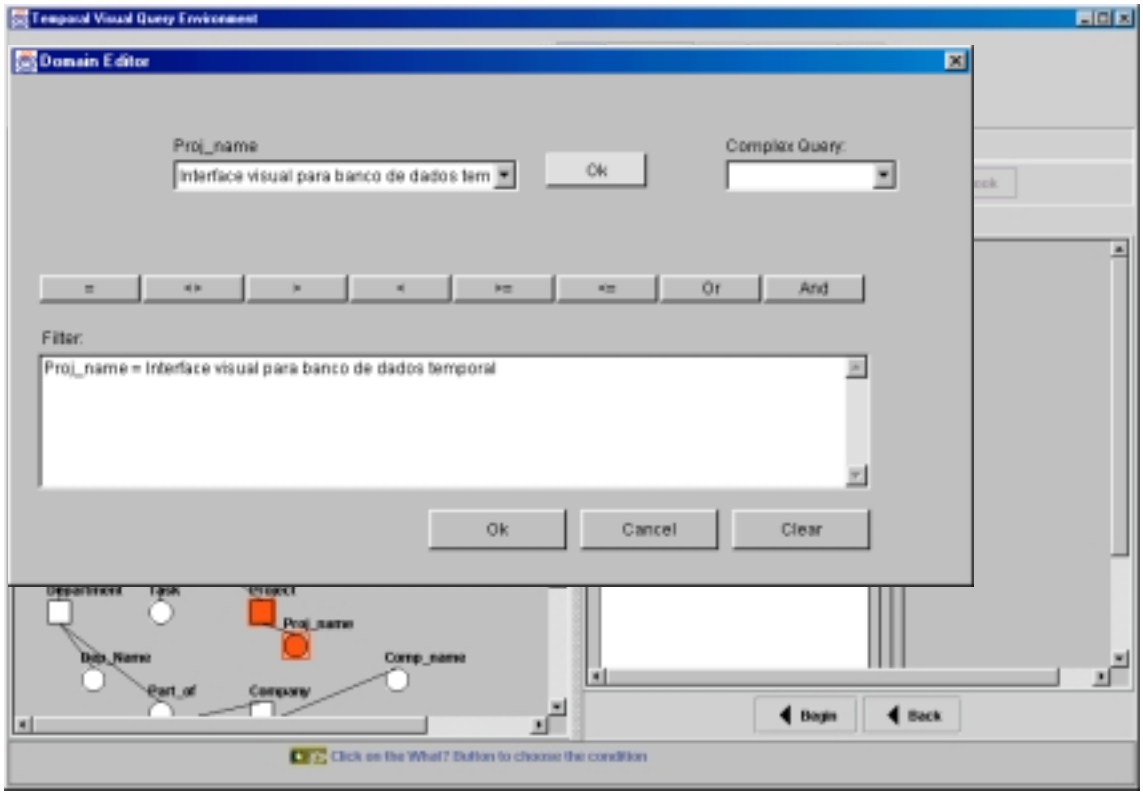


Figura 4.3 – Especificação da condição da consulta no TVQE

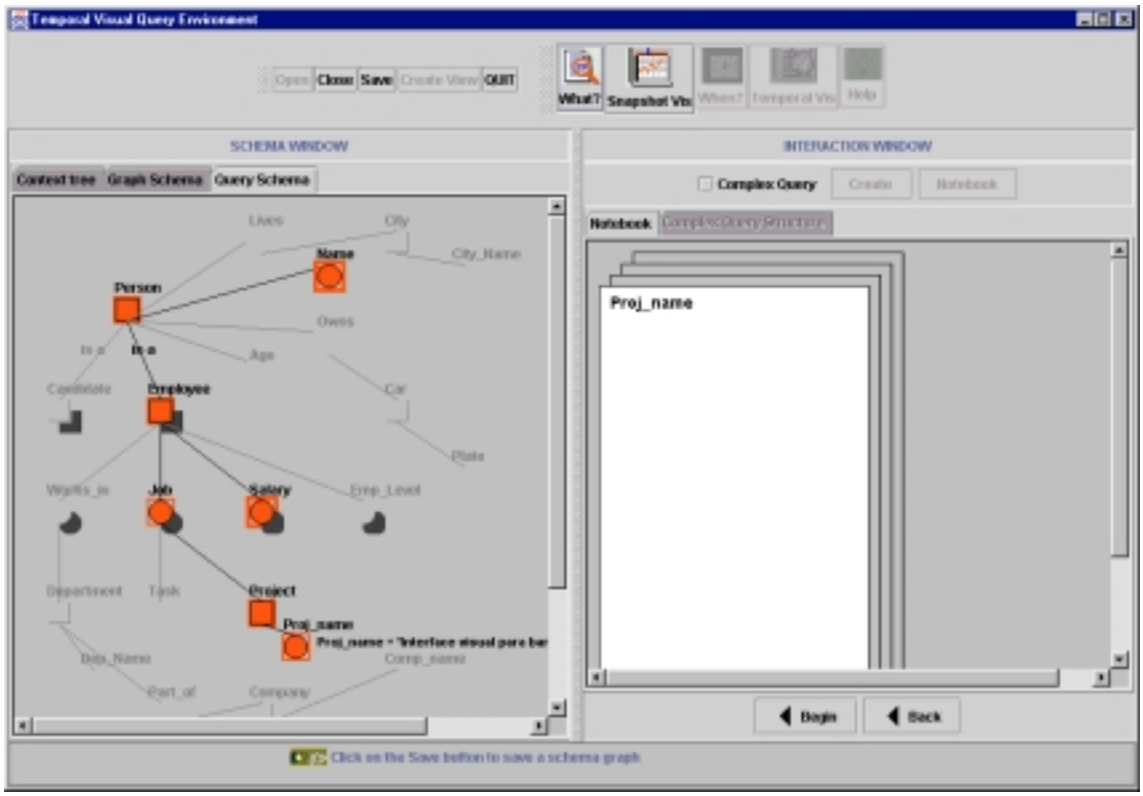


Figura 4.4 – Resultado intensional da consulta

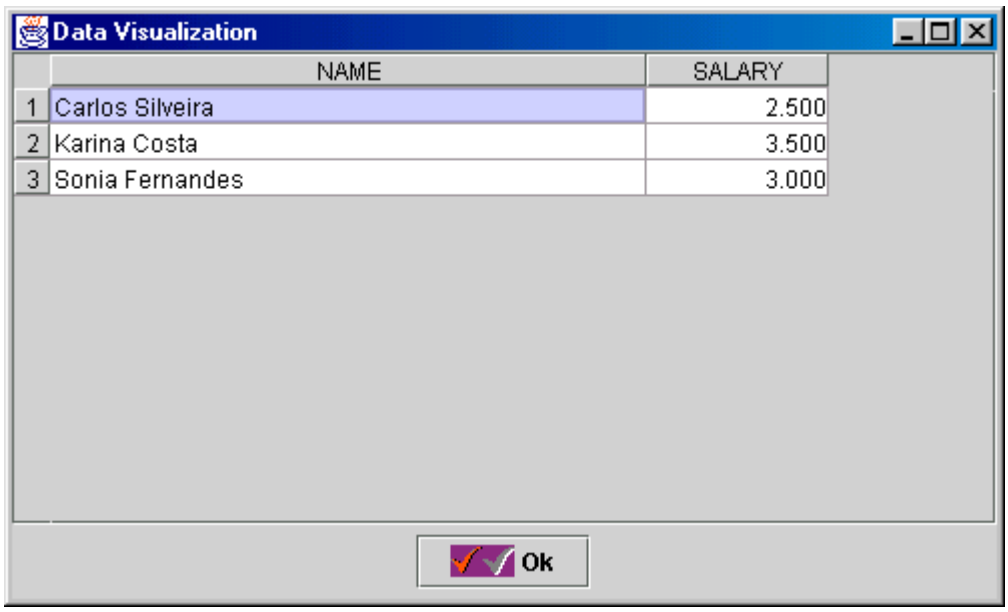


Figura 4.5 – Resultado extensional da consulta

O mapeamento dessa consulta para SQL, considerando o esquema de banco de dados descrito na seção 4.2 é o seguinte:

SELECT Pessoa.Nome, Empregado_Salario.Salario
FROM Pessoa, Empregado, Empregado_Salario, Trabalho, Projeto
WHERE **Empregado.de <= 20000816** and **Empregado.ate >= 20000816** and Empregado.Emp_Id = Pessoa.PessoaId and **Empregado_Salario.de <= 20000816** and **Empregado_Salario.ate >= 20000816** and Empregado_Salario.Emp_Id = Empregado.Emp_Id and **Trabalho.de <= 20000816** and **Trabalho.ate >= 20000816** and Trabalho.ProjetoId = Projeto.ProjetoId and Trabalho.Emp_Id = Pessoa.PessoaId and Nome_Proj = 'Interface visual para banco de dados temporal'

Note que todas as tabelas temporais relacionadas na consulta têm seus atributos temporais (**de** e **ate**) comparados com uma constante temporal **t**, que representa um instante de tempo atual, neste caso **t = 20000816**, onde **t** tem o seguinte formato: AAAAMMDD, com AAAA representando o ano, MM o mês e DD o dia.

4.3.3. Consultas Temporais

As consultas temporais são aquelas em que o usuário especifica condições temporais para seus atributos, relacionamentos ou classes temporais, permitindo assim a recuperação de dados passados, presentes e futuros do BD. Este tipo de consulta é expressa visualmente no TVQE quando o usuário formula sua consulta e ativa o ícone *Temporal Vis*, para visualização dos dados. As condições temporais são expressas quando o usuário ativa o ícone *When?*

Dentre as consultas temporais suportadas pelo TVQE, segundo a taxonomia descrita na seção 4.3.1, apenas aquelas que envolvem **seleção de dados/projeção temporal**, **seleção de dados/projeção mista** e **seleção temporal/projeção de dados** foram implementadas, pois as demais para serem formuladas no TVQE envolvem os operadores de referência e agregação temporal citados a seguir.

Apenas os operadores temporais *All History*, *Instant* e *Period* foram transformados em instruções SQL. Os operadores de referência temporal (*Temporal Reference To...*) e o operador de agregação temporal (*Temporal Aggregation*) não foram implementados, pois ainda não estão bem definidos na interface do ambiente TVQE. As consultas temporais no TVQE permitem o uso desses operadores separadamente, ou seja, apenas um operador por vez como parte de uma condição temporal.

A granularidade informada pelo usuário em uma condição temporal para um instante ou período de tempo, pode ser diferente da dos atributos temporais na tabela de tempo-válido presente no banco de dados. Se a granularidade informada pelo usuário for menos fina que da tabela de tempo-válido, por exemplo: o ano de *1999* tem granularidade menos fina que o dia (*1999/12/01*) informado pelo usuário, então a completamos com a devida combinação dos grânulos que fazem parte dos conjuntos de granularidades **Gmin** e **Gmax**, descritos a seguir, para que a mesma se iguale. Caso contrário, não fazemos nenhuma alteração. Considerando o exemplo anterior fazemos o seguinte: completamos o ano de *1999* com os grânulos mínimos ou máximos de mês e dia, neste caso, *01/01* ou *12/31*, respectivamente. Gerando assim a nova granularidade (dia): *1999/01/01* ou *1999/12/31*, para que esta possa ser comparada com *1999/12/01*.

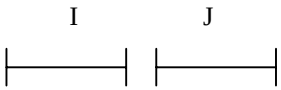
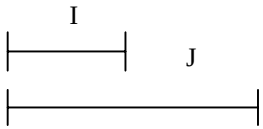
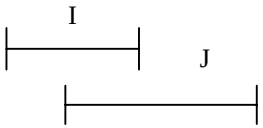
Os valores mínimos e máximos referentes aos grânulos são adicionados a granularidade informada pelo usuário baseado nos operadores de comparação (<, >, >=, <=, =) que fazem parte das condições dos operadores temporais, de modo que não haja perda na semântica destes operadores temporais.

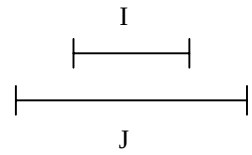
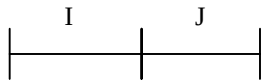
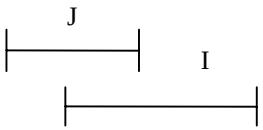
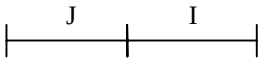
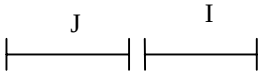
Seja **Tab** = {$tabela_1$, ..., $tabela_n$}, com n >=1, o conjunto de tabelas temporais e/ou não temporais, **Atr** = {$atributo_1$, ..., $atributo_n$}, com n >= 1, o conjunto de atributos temporais e/ou não temporais pertencentes a Tab, **Gmin** = {01, 01, 00} o conjunto das granularidades mínimas, onde o primeiro elemento de *Gmin* representa o primeiro mês do ano, o segundo o primeiro dia do mês e o terceiro a primeira hora do dia; e **Gmax** = {12, diaDomes, 23} o conjunto de granularidades máximas, onde o primeiro elemento representa o último mês do ano, o segundo o último dia do mês, que varia de 28 a 31 dependendo do mês informado pelo usuário, e o terceiro representa a última hora do dia. O ano não faz parte de *Gmin* e nem de *Gmax*, por ser a granularidade mínima informada pelo usuário. A tabela 4.5 ilustra como cada dos operadores temporais foram transformados em instruções SQL de acordo com a variação da granularidade, considerando os conjuntos acima descritos.

Operador	**Mapeamento para SQL, quando a granularidade informada pelo usuário é mais fina ou igual a dos atributos temporais (*de* e *ate*)**	**Mapeamento para SQL, quando a granularidade informada pelo usuário é menos fina que a dos atributos temporais (*de* e *ate*)**
ALL HISTORY	SELECT Atr FROM Tab	
First Interval	SELECT Atr FROM Tab WHERE **$tabela_T$.de IN (SELECT MIN(de) FROM $tabela_T$)** , onde $tabela_T$ é um tabela temporal ∈ Tab	
Specified Interval[8]	SELECT Atr FROM Tab ORDER BY **$tabela_T$.de, $tabela_T$.ate**, onde $tabela_T$ é uma tabela temporal∈ Tab	
Last Interval	SELECT Atr FROM Tab WHERE **$tabela_T$.de IN (SELECT MAX(de) FROM $tabela_T$)**, onde $tabela_T$ é uma tabela temporal ∈ Tab	

[8] Note que o mapeamento para SQL recupera o histórico de dados ordenado pelos atributos temporais (**de** e **ate**). Portanto, os dados são visualizados de forma ordenada possibilitando ao usuário identificar o intervalo de tempo desejado. Não sendo possível faze-lo diretamente em SQL.

INSTANT (T)	Onde T é um instante de tempo.	
Begin (T)	SELECT Atr FROM tab WHERE **$tabela_T$.de = T**, onde $tabela_T$ é uma tabela temporal∈ Tab	SELECT Atr FROM tab WHERE **$tabela_T$.de >= TGmim AND $tabela_T$.de <= TGmax**
At (T)	SELECT Atr FROM tab WHERE **$tabela_T$.de <= T AND $tabela_T$.ate>= T**, onde $tabela_T$ é uma tabela temporal∈ Tab	SELECT Atr FROM tab WHERE **($tabela_T$.de < TGmin OR ($tabela_T$.de >= TGmin AND $tabela_T$.de <= TGmax)) AND ($tabela_T$.ate > TGmax OR ($tabela_T$.ate >= TGmin AND $tabela_T$.ate <= TGmax))**
End (T)	SELECT Atr FROM Tab WHERE **$tabela_T$.ate = T**, onde $tabela_T$ é uma tabela temporal∈ Tab	SELECT Atr FROM Tab WHERE **$tabela_T$.ate >= TGmin AND $tabela_T$.ate <= TGmax**

PERIOD (I, J)	Onde *I* e *J* são dois intervalos de tempo. O *I* representa o intervalo presente no BD cujo início é representado por I_1 e fim por I_2, e eqüivalem aos atributos temporais **de** e **ate**, respectivamente**.** O *J* representa o intervalo especificado pelo usuário cujo início é representado por J_1 e fim por J_2.	
Before (I, J) 	SELECT Atr FROM Tab WHERE $\mathbf{I_2 < J_1}$	SELECT Atr FROM Tab WHERE $\mathbf{I_2 < J_1 Gmin}$
Start (I, J) 	SELECT Atr FROM Tab WHERE $\mathbf{I_1 = J_1}$ **AND** $\mathbf{I_2 < J_2}$	SELECT Atr FROM Tab WHERE $\mathbf{I_1 >= J_1 Gmin}$ **AND** $\mathbf{I_1 <= J_1 Gmax}$ **AND** $\mathbf{I_2 < J_2}$
Cross (I, J) 	SELECT Atr FROM Tab WHERE $\mathbf{J_1 > I_1}$ **AND** $\mathbf{I_2 > J_1}$ **AND** $\mathbf{J_2 > I_2}$	SELECT Atr FROM Tab WHERE $\mathbf{J_1}$Gmin > $\mathbf{I_1}$ **AND** $\mathbf{I_2 > J_1}$ Gmax **AND** $\mathbf{J_2}$ Gmin > $\mathbf{I_2}$

During (I, J)  	SELECT Atr FROM Tab WHERE $\mathbf{I_1 > J_1}$ **AND** $\mathbf{I_2 < J_2}$	SELECT Atr FROM Tab WHERE $\mathbf{I_1 > J_1Gmax}$ **AND** $\mathbf{I_2 < J_2Gmin}$
Meet (I, J)  	SELECT Atr FROM Tab WHERE $\mathbf{I_2 = J_1}$	SELECT Atr FROM Tab WHERE $\mathbf{I_2 >= J_1Gmin}$ **AND** $\mathbf{I_2 <= J_1Gmax}$
Cross (J, I)  	SELECT Atr FROM Tab WHERE $\mathbf{I_1 > J_1}$ **AND** $\mathbf{J_2 > I_1}$ **AND** $\mathbf{I_2 > J_2}$	SELECT Atr FROM Tab WHERE $\mathbf{I_1 > J_1Gmax}$ **AND** $\mathbf{J_2Gmin > I_1}$ **AND** $\mathbf{I_2 > J_2Gmax}$
Follow (I, J)  	SELECT Atr FROM Tab WHERE $\mathbf{J_2 = I_1}$	SELECT Atr FROM Tab WHERE $\mathbf{I_1 >= J_2Gmin}$ **AND** $\mathbf{I_1 <= J_2Gmax}$
After (I, J)  	SELECT Atr FROM Tab WHERE $\mathbf{J_2 < I_1}$	SELECT Atr FROM Tab WHERE $\mathbf{J_2Gmax < I_1}$

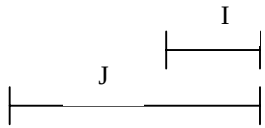
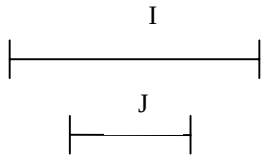
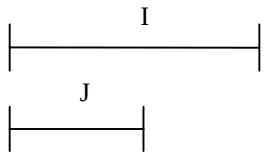
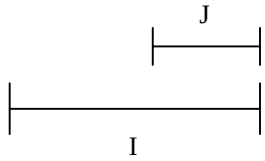
Finish (I, J) 	SELECT Atr FROM Tab WHERE **$I_2 = J_2$ AND $I_1 > J_1$**	SELECT Atr FROM Tab WHERE **$I_2 >= J_2$Gmin AND $I_2 <= J_2$Gmax AND $I_1 > J_1$Gmax**
Contain (I, J) 	SELECT Atr FROM Tab WHERE **$I_1 < J_1$ AND $I_2 > J_2$**	SELECT Atr FROM Tab WHERE **$I_1 < J_1$Gmin AND $I_2 > J_2$Gmax**
Started-by (I, J) 	SELECT Atr FROM Tab WHERE **$I_1 = J_1$ AND $I_2 > J_2$**	SELECT Atr FROM Tab WHERE **$I_1 >= J_1$Gmin AND $I_1 <= J_1$Gmax AND $I_2 > J_2$Gmax**
Finished-by (I, J) 	SELECT Atr FROM Tab WHERE **$I_1 < J_1$ AND $I_2 = J_2$**	SELECT Atr FROM Tab WHERE **$I_1 < J_1$Gmin AND $I_2 >= J_2$Gmin AND $I_2 <= J_2$Gmax**

Tabela 4.5 – Mapeamento dos operadores temporais para SQL

A seguir exemplificaremos algumas consultas temporais no TVQE utilizando alguns dos operadores temporais acima descritos. Estas consultas estão relacionadas com o esquema de agência de empregos visualizado no TVQE.

- **Operador *All History***

Como exemplo de uma consulta que utiliza o operador *All History*, temos a seguinte: *Qual o histórico salarial dos empregados?* Para formularmos esta consulta no TVQE, fazemos o seguinte: selecionamos na agenda "gráfica" os índices contexto-pessoal *(personal-context)*, pessoa *(person)*, empregado *(employee)*, nome do empregado *(name)* e salário *(salary)*. Os índices da agenda que representam salário *(salary)* e nome *(name)* ficam no estado visualizado (pintados e contornados de vermelho), pois irão fazer parte da cláusula SELECT, quando transformados para SQL. Para formular a condição temporal temos que selecionar o atributo *salary* e ativar o ícone de condições temporais *When?*, para então selecionarmos a condição *All History* que recuperará todo o histórico de dados relacionado ao atributo *salary*. A figura 4.6 ilustra a especificação desta condição no TVQE.

Figura 4.6 – Especificação da condição temporal *All History* para o atributo *Salary*

As figuras 4.7 e 4.8 ilustram o resultado intensional e extensional da consulta, respectivamente. Para filtrar os intervalos de tempo recuperados pela consulta é só selecionar uma das opções a seguir.

1. *First Interval* – Recupera o intervalo de tempo de cada objeto (empregado), cujo início (*begin_time*) é o menor, isto é, o primeiro intervalo. Neste caso, temos as tuplas 1, 3, 4 e 6, veja figura 4.8;
2. *Specified Interval* – Recupera o n-ésimo intervalo especificado pelo usuário para cada objeto (empregado), cujo início (*begin_time*) é o menor, isto é, o n-ésimo menor intervalo. Por exemplo, gostaríamos de recuperar o terceiro intervalo de cada objeto (empregado), neste caso, não temos nenhum dado recuperado, pois não existe o terceiro intervalo para cada objeto listado na figura 4.8;
3. *Last Interval* – Recupera o intervalo de tempo de cada objeto (empregado), cujo início (*begin_time*) é o maior, isto é, o último intervalo. Neste caso, temos as tuplas 2, 3, 5 e 7, veja figura 4.8.

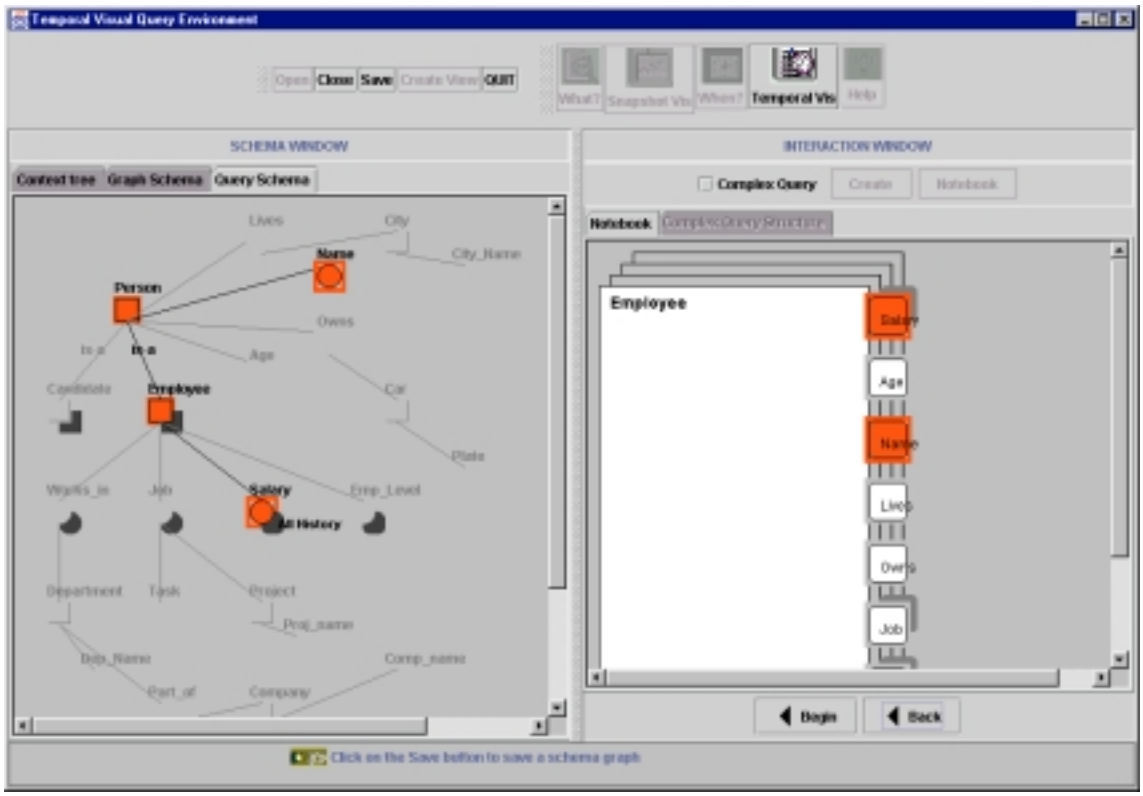


Figura 4.7 – Resultado intensional da consulta para o operador *All History*

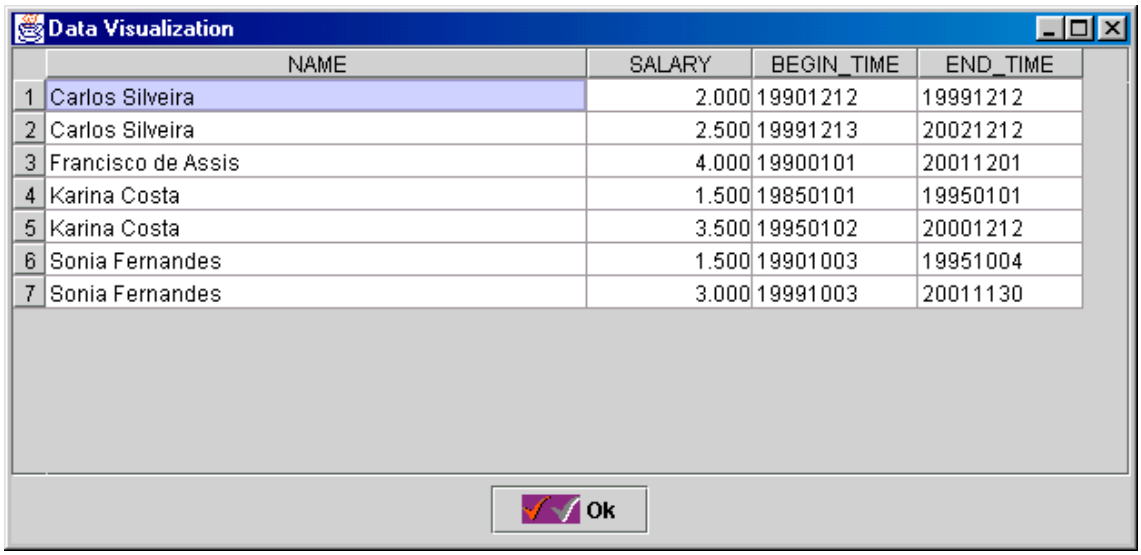


Figura 4.8 – Resultado extensional da consulta para o operador *All History*

Gostaríamos de ressaltar que não foi possível transformar para SQL as consultas que utilizam os operadores de filtragem *First Interval*, *Specified Interval* e *Last Interval*, considerando a filtragem sob cada objeto. No entanto, é possível se considerarmos todo o conjunto de objetos. Logo, temos os seguintes resultados:

1. *First Interval* – Recupera o intervalo de tempo, cujo início (*begin_time*) é o menor de todos, isto é, o primeiro intervalo. Neste caso, temos a tupla número quatro (empregada *Karina, salário 1.500*), veja figura 4.8;
2. *Specified Interval* – Recupera o n-ésimo intervalo de tempo especificado pelo usuário, cujo início (*begin_time*) é o menor de todos. Por exemplo, gostaríamos de recuperar o terceiro intervalo, que neste caso, corresponde à tupla número seis (empregada *Sonia Fernandes, salário 1.500*), veja figura 4.8;
3. *Last Interval* - Recupera o intervalo de tempo, cujo início (*begin_time*) é o maior de todos, isto é, o último intervalo. Neste caso, temos a tupla número dois (empregado *Carlos Silveira, salário 2.500*), veja figura 4.8.

O mapeamento destas consultas para SQL considerando os operadores temporais *All History*, *First Interval*, *Specified Interval* e *Last Interval*, a partir do esquema de banco de dados descrito na seção 4.2, está ilustrado seguir.

1. Mapeamento para o operador *All History*

```
SELECT DISTINCT Pessoa.Nome, Empregado_Salario.Salario,
Empregado_Salario.de, Empregado_Salario.ate
FROM Pessoa, Empregado, Empregado_Salario
WHERE Empregado.Emp_Id = Pessoa.PessoaId and
Empregado_Salario.Emp_Id=Empregado.Emp_Id
```

2. Mapeamento para o operador *First Interval*

```
SELECT DISTINCT Pessoa.Nome, Empregado_Salario.Salario,
Empregado_Salario.de, Empregado_Salario.ate
FROM Pessoa, Empregado, Empregado_Salario
WHERE Empregado.Emp_Id = Pessoa.PessoaId and
Empregado_Salario.Emp_Id=Empregado.Emp_Id and Empregado_Salario.de in
(SELECT MIN (Empregado_Salario.de) FROM Empregado_Salario)
```

3. Mapeamento para o operador *Specified Interval*

```
SELECT DISTINCT Pessoa.Nome, Empregado_Salario.Salario,
Empregado_Salario.de, Empregado_Salario.ate
FROM Pessoa, Empregado, Empregado_Salario
WHERE Empregado.Emp_Id = Pessoa.PessoaId and
Empregado_Salario.Emp_Id=Empregado.Emp_Id
ORDER BY Empregado_Salario.de, Empregado_Salario.ate
```

4. Mapeamento para o operador *Last Interval*

```
SELECT DISTINCT Pessoa.Nome, Empregado_Salario.Salario,
Empregado_Salario.de, Empregado_Salario.ate
FROM Pessoa, Empregado, Empregado_Salario
WHERE Empregado.Emp_Id = Pessoa.PessoaId and
Empregado_Salario.Emp_Id=Empregado.Emp_Id and Empregado_Salario.de in
(SELECT MAX (Empregado_Salario.de) FROM Empregado_Salario)
```

- Operador *Instant*

Como exemplo de uma consulta que utiliza o operador *Instant*, temos a seguinte: *Quais foram os salários dos empregados em 10/01/1998?* O processo de formulação da consulta segue os mesmos passos realizados na consulta anterior com o operador *All History*, a diferença está na condição temporal, ou seja, o operador *Instant*. Para especificar a condição selecionamos a opção *Instant* na janela *When?*, escolhemos a granularidade temporal, neste caso ano, mês e dia, selecionamos seus valores (1998, 01, 10) e depois o operador ***At***, pois este operador recupera dados temporais cujo instante de tempo **t** está dentro dos intervalos recuperados, ou seja, queremos os salários no instante de tempo **t = 10/01/1998**. A figura 4.9 ilustra a especificação da condição da consulta no TVQE e as figuras 4.10(a) e 4.10(b) o resultado intensional e extensional da consulta, respectivamente.

Figura 4.9 – Especificação da condição da consulta para o operador *At*

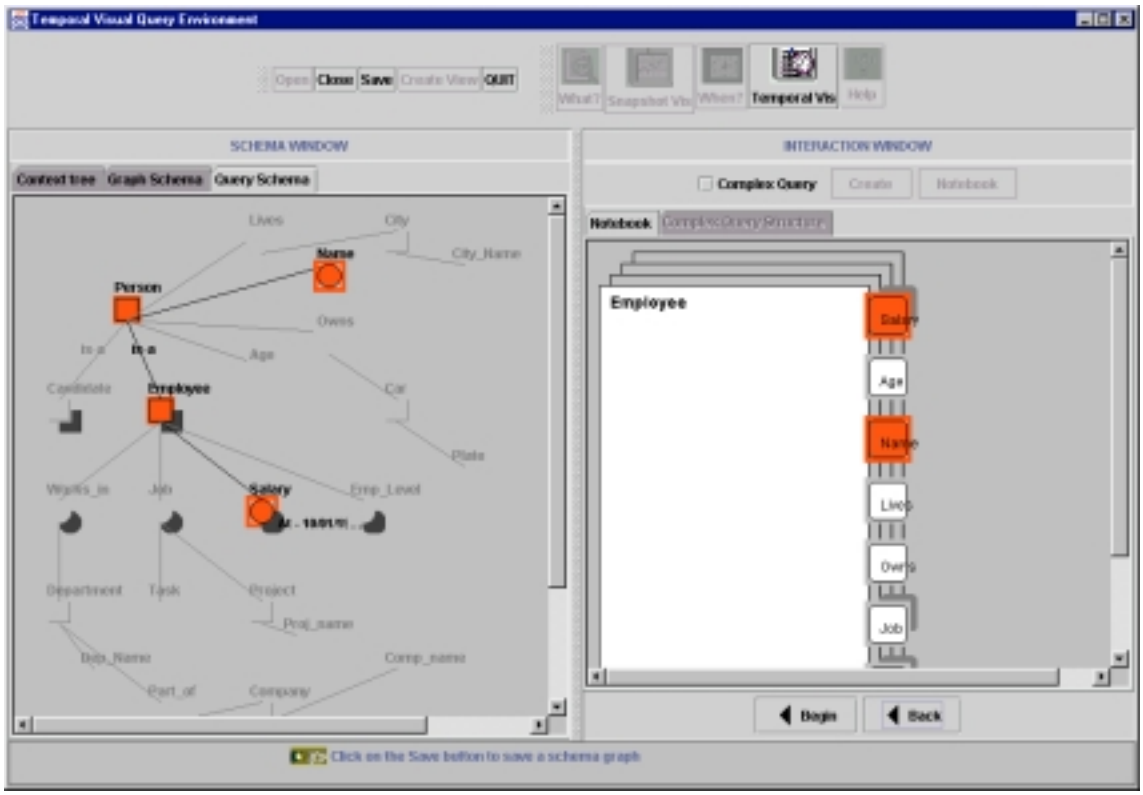


Figura 4.10(a) – Resultado intensional da consulta para o operador *At*

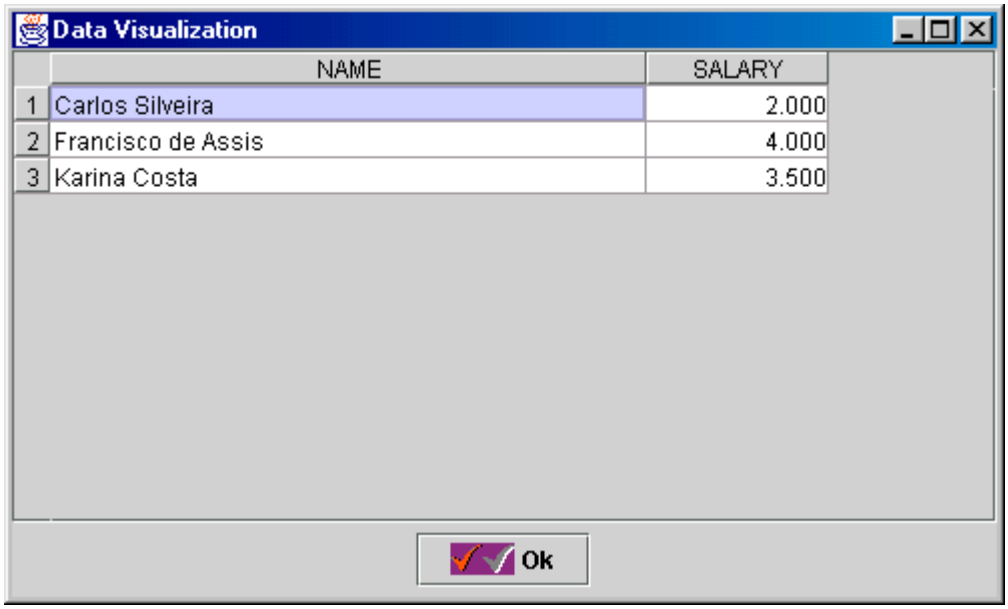


Figura 4.10(b) – Resultado extensional da consulta para o operador *At*

O mapeamento desta consulta para SQL, considerando o esquema de banco de dados descrito na seção 4.2 é o seguinte:

SELECT DISTINCT Pessoa.Nome, Empregado_Salario.Salario
FROM Pessoa, Empregado, Empregado_Salario
WHERE Empregado.Emp_Id=Pessoa.PessoaId and
Empregado_Salario.de<=19990110 and Empregado_Salario.ate>=19990110 and
Empregado_Salario.Emp_Id=Empregado.Emp_Id

Note que apenas os atributos temporais das relações de tempo-válido são comparados com constante(s) temporal(ais). Como a condição temporal foi sob o atributo salário *(Salary)*, então obtemos sua estrutura relacional equivalente no BD, denominada **Empregado_Salario**, que é uma relação de tempo-válido cujos atributos temporais são comparados ao valor especificado pelo usuário para o operador *At*, neste caso 10/01/1998. Como a granularidade dos atributos temporais da tabela *Empregado_Salario* é igual ao do instante de tempo **T** informado pelo usuário, ou seja, dia. Então, o operador *At* recuperará os dados cujos valores dos atributos temporais (**de** e **ate**) satisfaçam a seguinte condição: **Empregado_Salario.de <= T AND Empregado_Salario.ate >= T**, onde **T = 10/01/1998**.

- **Operador *Period***

Com este operador podemos especificar um período na consulta que será relacionado aos períodos dos objetos no banco de dados. Para descrever o operador *Period* utilizamos o seguinte exemplo: *Quais foram os projetos em que Francisco de Assis trabalhou durante o período de 1990 até 2000?* Como já conhecemos o processo de formulação da consulta no TVQE, vamos direto à parte de especificação da condição da consulta, neste caso, selecionamos o índice da agenda sobre o qual atua a condição temporal, ou seja, o índice que representa o relacionamento Trabalho *(Job)*, logo após ativamos o ícone *When?* e selecionamos a opção *Period* e o operador ***During***, pois gostaríamos de saber os projetos que *Francisco* trabalhou durante o período especificado. A figura 4.11 ilustra a especificação da condição desta consulta e as

figuras 4.12(a) e 4.12(b) o resultado intensional e extensional da consulta, respectivamente.

Figura 4.11 – Especificação da condição da consulta para o operador *During*

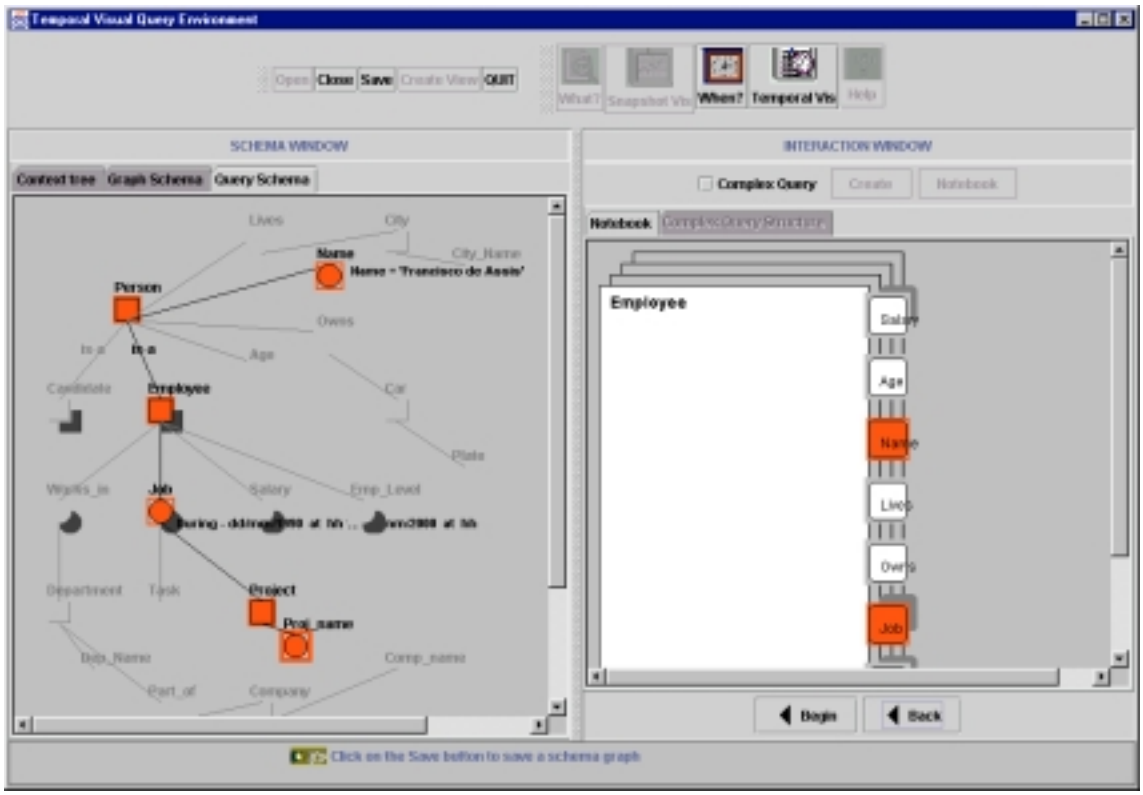


Figura 4.12(a) – Resultado intensional da consulta usando o operador *During*

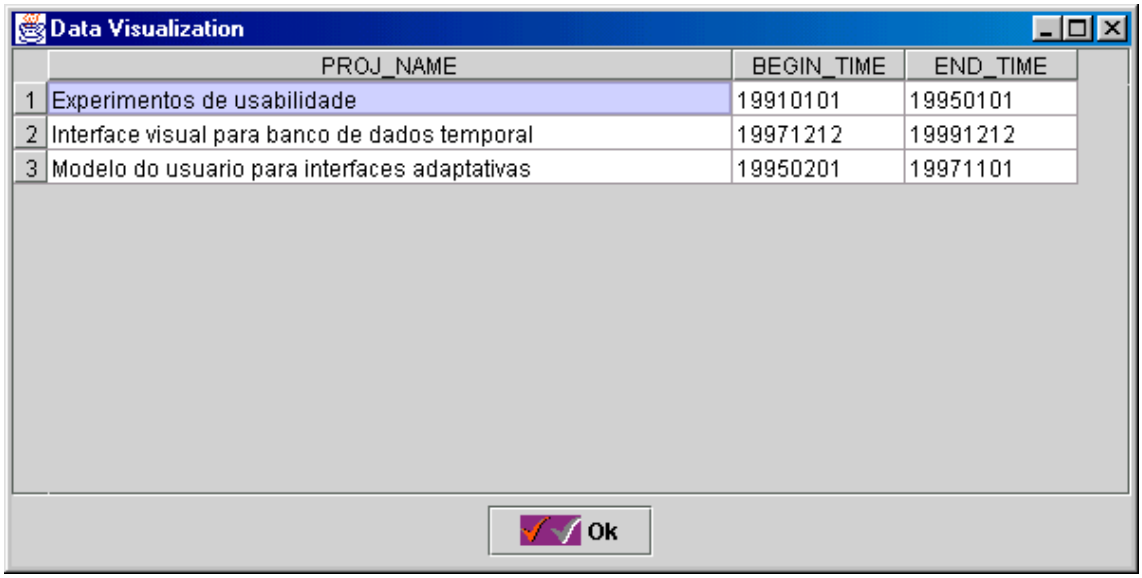


Figura 4.12(b) – Resultado extensional da consulta usando o operador *During*

O mapeamento desta consulta para SQL, considerando o esquema de banco de dados descrito na seção 4.2 é o seguinte:

SELECT DISTINCT Projeto.Proj_Nome, Trabalho.de, Trabalho.ate
FROM Pessoa, Empregado, Trabalho, Projeto
WHERE Empregado.Emp_Id=Pessoa.PessoaId and Trabalho**.de > 19901231 and**
Trabalho**.ate < 20000101** and Trabalho.ProjetoId=Projeto.ProjetoId and
Trabalho.Emp_Id=Empregado.Emp_Id and Nome='Francisco de Assis'

Note que apenas os atributos temporais das relações de tempo-válido são comparados com constante(s) temporal(ais). Como a condição temporal foi sob o relacionamento *Job* (Trabalho), então obtemos sua estrutura relacional equivalente no BD, denominada **Trabalho**, que é uma relação de tempo-válido cujos atributos temporais são comparados ao valor especificado pelo usuário para o operador *During*, neste caso temos o período de 1990 a 2000. Como a granularidade dos atributos temporais (dia) da tabela *Trabalho* é mais fina que do período de tempo **J** informado pelo usuário (ano). Então, o operador *During* recuperará os dados cujos valores dos atributos temporais (**de** e **ate**) satisfaçam a seguinte condição: **Trabalho.de > J_1Gmax and Trabalho.ate < J_2Gmin,** onde J_1=1990, J_2=2000, Gmin = 0101 e Gmax = 1231.

4.4. Conclusão

Neste capítulo apresentamos a especificação e implementação de um componente que faz parte da arquitetura do ambiente TVQE, denominado Tradutor. O Tradutor realiza o mapeamento de consultas realizadas no TVQE para uma linguagem textual de consulta, no caso SQL. A especificação baseia-se no mapeamento do esquema TGM (esquema gráfico do TVQE) para o modelo relacional. Este mapeamento é equivalente ao mapeamento do diagrama de entidade e relacionamento (DER) para o relacional.

A partir da estrutura TGM-Relação gerada através do mapeamento, que relaciona cada componente do esquema TGM com um da estrutura relacional, o Tradutor lê o esquema de consulta gerado pelo usuário e obtém sua estrutura relacional correspondente, gerando portanto a consulta na linguagem SQL que será executada por um SGBD relacional no qual está o banco de dados em questão. A estrutura TGM-Relação foi gerada "manualmente" para que testássemos o Tradutor, pois a sua implementação não faz parte do escopo dessa dissertação. Esta estrutura é representada internamente por um arquivo no formato "**.db**". Para o esquema da agência de empregos criamos o arquivo *employeement.db*.

A forma de como representar o resultado extensional da consulta no TVQE também não faz parte do escopo dessa dissertação, portanto a visualização desse resultado através de uma forma tabular, onde uma tabela é visualizada contendo o resultado da consulta, serve apenas como meio para validar os resultados das consultas realizadas pelo usuário, pois sabemos que uma consulta temporal gera muitos dados como resultado dependendo do tamanho do banco de dados, mas como o nosso banco de dados de teste é relativamente pequeno, não teve problema.

Foram realizados vários testes para validação da aplicação (Tradutor), onde a partir da consulta gerada no TVQE pelo usuário, esta era mapeada para a linguagem SQL e então executada. Os SGBDs utilizados foram o Oracle8i e o Interbase 5.0, nos quais foram criados um banco de dados, neste caso o banco de dados de uma agência de empregos, denominado AgenciaEmpregos. Este banco de dados tem seu *script* de criação das tabelas ilustrado no apêndice B e seu esquema na seção 4.2.

Capítulo 5 - Aspectos de Implementação

Um aspecto importante a ser considerado no processo de implementação é o uso de uma ferramenta adequada de implementação. Como o ambiente TVQE foi implementado utilizando a linguagem de programação Java e o Tradutor é um dos módulos presentes no TVQE, utilizamos a mesma linguagem para sua implementação.

Neste capítulo descrevemos os aspectos de implementação relacionados ao Tradutor e ao *applet TVQE*, considerando as principais características presentes na linguagem Java, o componente Java de conexão com banco de dados (JDBC), *applets Java* e as estruturas de classes utilizadas na especificação da aplicação.

5.1. Características da Linguagem Java

As principais características da linguagem Java são as seguintes:

- ***Simples***, Java foi projetada de forma a permitir um rápido aprendizado pelos programadores, assim ela possui um pequeno número de construtores. Outro objetivo de seu projeto foi torná-la familiar para a maioria dos programadores, de forma a facilitar a migração, daí sua semelhança com C/C++;
- ***Orientada a Objeto***, Java é uma linguagem de programação orientada a objeto. Isto significa que com ela você se concentra nos objetos de sua aplicação e nos métodos que manipulam aqueles objetos. Portanto, Java dispõe de todas as características da orientação a objeto, tais como: os conceitos de classe, objeto, polimorfismo, herança, encapsulamento, entre outros;
- ***Distribuída***, Java foi projetada para suportar aplicações de rede, ou seja, é uma linguagem distribuída. Suporta vários níveis de conexões de rede que

são oferecidos através de classes dentro do pacote *java.net*, contido no *Java Developers Kit* (JDK). Dentro desse pacote pode-se encontrar a classe *URL*, para abrir e acessar objetos remotos na Internet, entre outras, sendo muito úteis para a criação de aplicações cliente/servidor;

- ***Interpretada***, os compiladores Java geram um código intermediário, denominado *byte-code*, este código é independente de máquina. Para rodar um programa em Java usa-se um interpretador Java (construído especialmente para a arquitetura de máquina que está sendo utilizado) para executar os *byte-codes* gerados pelo compilador. Este interpretador, traduz os *byte-codes* para instruções de máquina do seu processador. O interpretador, em conjunto com o sistema *run-time* Java é denominado, Máquina Virtual Java;
- ***Robusta***, Java foi projetada para escrever código robusto ou altamente confiável. Certamente Java não elimina a necessidade da garantia da qualidade do *software*, mas oferece subsídios para construção de *software* confiável. Entre as características que a tornam robusta podemos citar sua tipagem, oferecida pelo seu compilador, e que exige a conversão explícita entre os tipos em Java;
- ***Segura***, este aspecto é especialmente importante por Java ser uma linguagem de natureza distribuída. Sem a garantia de segurança, certamente não poderíamos carregar programas através da rede e executá-los em nosso computador. Java foi projetada para ser segura, prover vários níveis de controle de segurança que a protegem contra códigos maliciosos;
- ***Arquitetura Neutra***, os códigos fontes dos programas Java são compilados para um formato de arquitetura neutra, denominado *byte-code*. Sendo assim, uma vez escrito e compilado um programa Java estará habilitado a rodar em qualquer sistema que disponha de uma máquina virtual Java;
- ***Portável***, ser interpretada e ter arquitetura neutra já traz implicitamente a portabilidade da linguagem Java mas, mesmo assim, ela garante em alguns aspectos não existir dependência de implementação da especificação da linguagem. Por exemplo, Java especifica explicitamente o tamanho de cada tipo de dado primitivo, assim como seu comportamento aritmético;

- ***Performance***, por ser uma linguagem interpretada, Java não é uma linguagem tão rápida em relação às compiladas, como C. Em comparação com tais linguagens, por exemplo, ela pode chegar a ser até 20 vezes mais lenta. Entretanto, a performance de Java chega a ser aceitável, para muitas aplicações, tais como, aplicações baseadas em rede e/ou com interfaces gráficas, uma vez que em muitas destas aplicações a maior parte do tempo que se perde é esperando alguma entrada/saída do usuário, ou algum dado sendo carregado pela rede;
- ***Multithread*** *(Concorrente)*, em Java, os programadores têm a liberdade de criar dentro de um mesmo programa várias linhas de execução. Sendo assim, quando um programa Java concorrente roda, ele pode estar realizando várias tarefas "ao mesmo tempo". Além disso, você pode controlar estas linhas (*threads*) estabelecendo ordens de prioridade para as mesmas, e sincronizando o acesso a determinados dados.

5.2. Conexão com o Banco de Dados

Baseado na arquitetura funcional do TVQE descrita no início do capítulo 4, sabemos que após o Tradutor realizar o mapeamento da consulta para SQL, esta é enviada para ser processada por um SGBD através do dispositivo JDBC que faz a conexão com o banco de dados. Portanto, vamos entender um pouco como este componente de conexão com banco de dados funciona.

Com o objetivo de desenvolver aplicações Java de acesso a bancos de dados, cujo código fosse independente de um SGBD específico ou de um mecanismo de conexão com o banco de dados, decidiu-se então definir uma estrutura genérica de acesso a bancos de dados que provê uma interface uniforme no topo de uma variedade de diferentes módulos de conexão com o banco de dados. Portanto, foi definida uma API comum de baixo nível que suporta funcionalidades básicas da linguagem SQL, denominada API JDBC (*Java Database Connective*). Esta API, em troca, permite desenvolver ferramentas de alto de nível de acesso a bancos de dados e APIs [9].

A API JDBC é formada por uma série de interfaces Java que deixam o programador de aplicação abrir conexões para bancos de dados particulares, executar declarações SQL, e processar resultados [9]. As interfaces mais importantes são:

- *java.sql.DriverManager*, manipula o carregamento de *drivers* e provê suporte para criar novos banco de dados;
- *java.sql.Connection*, representa uma conexão para um banco de dados particular;
- *java.sql.Statement*, atua como um repositório, no qual declarações SQL são executadas em uma dada conexão;
- *java.sql.ResultSet*, controla o acesso às tuplas da tabela que apresenta o resultado de uma consulta SQL.

Existem várias alternativas para implementação de JDBC [13]. A adotada pelo Tradutor foi a ponte JDBC-ODBC, por atender às necessidades do momento, mas futuramente migraremos para o JDBC puro. Estas alternativas estão listadas abaixo.

- ***JDBC-ODBC bridge***, é um *driver* que implementa operações JDBC traduzindo-as para operações ODBC para qualquer banco de dados que tenha o *driver* ODBC disponível. Note que o código binário ODBC, e em muitos casos o código cliente do banco de dados, deve ser carregado em cada máquina cliente que utiliza este *driver*. Portanto, este tipo de *driver* é mais apropriado para redes corporativas onde não há maiores problemas com as instalações clientes, ou para código de servidor de aplicação escrito em Java em uma arquitetura de três camadas (*three-tier*);
- ***Native-API partly-Java***, é um *driver* que converte as chamadas JDBC para chamadas na API cliente do Oracle, Sybase, Informix, ou outros SGBDs. Note que, como no *JDBC-ODBC bridge*, também requer que alguns códigos binários sejam carregados em cada máquina cliente;
- O *driver* ***JDBC-net***, traduz as chamadas JDBC para um protocolo de rede genérico, independente de um SGBD, que é depois traduzido por um servidor para um protocolo de SGBD específico. Este servidor de rede está habilitado para conectar seus clientes Java com vários bancos de dados. Em geral, esta é a alternativa de implementação do JDBC mais flexível;
- ***Native-protocol pure Java***, é um *driver* que converte diretamente as chamadas JDBC para os protocolos de rede usados pelos SGBDs. Isto permite uma chamada direta da máquina cliente para o servidor do SGBD.

Supõe-se que os dois últimos tipos de *drivers* são os mais utilizados para acesso a banco de dados através do JDBC, pois os dois primeiros são apenas soluções interinas, onde os *drivers* de Java puro ainda não estão disponíveis. Para obter maiores informações sobre a API JDBC, veja [9][13].

5.3. *Applets* Java

Um *applet* é na realidade uma mini-aplicação. Podemos defini-lo como uma subclasse de *Panel*[9], usada para construir um programa para ser embutido em uma página HTML e executar em um navegador (*browser*) ou *applet viewer*[10], portanto pode conter componentes, mas não pode ter bordas ou legendas. Sob o ponto de vista de um programador, uma das grandes diferenças entre um *applet* e uma aplicação Java independente (*stand-alone*) é a ausência do método *main()*[11].

A tarefa de escrever um *applet* consiste na definição dos métodos que irá utilizar. A maioria destes métodos é definida pela classe *Applet* e alguns deles estão listados abaixo.

- *init ()* – Este método é chamado quando o *applet* é carregado no navegador ou *applet viewer*, sendo responsável por sua inicialização;
- *destroy ()* – Este método é chamado quando o *applet* é descarregado do navegador ou *applet viewer*. Sua função é liberar alguns recursos, como por exemplo, a memória em que estava alocado;
- *start ()* – Este método é chamado quando o *applet* se torna visível e começa a executar, sendo chamado depois do método *init()* e a cada momento que o *applet* é recarregado na página;
- *stop ()* – Este método é chamado pelo navegador ou *applet viewer* para parar a execução do *applet*.

[9] *Panel* é uma classe Java usada como repositório de outras classes [12].

[10] O *applet viewer* é um programa que lê ou carrega um ou mais documentos HTML especificado(s) por uma URL ou arquivo na linha de comando. Ele lê ou carrega todos os *applets* referenciados em cada documento e executa-os. Se nenhum dos documentos tiver a tag <APPLET>, o *applet viewer* não faz nada.

[11] É um método que representa o ponto de entrada da aplicação, ou seja, onde o programa começa a executar.

- Como os applets funcionam?

Os *applets* são armazenados em um servidor Internet/Intranet e carregados em múltiplas plataformas clientes, onde são executados por uma Máquina Virtual Java provida pelo navegador em uma máquina cliente. Quando o navegador encontra uma página na rede com um *applet*, a sua máquina virtual Java é inicializada e informações sobre a localização do *applet* através da *tag* <*APPLET*> são fornecidas para a mesma. Esta classe é então carregada dentro da máquina virtual Java para ver quais classes são necessárias para execução do *applet*.

As restrições de segurança existentes nos *applets* são necessárias para evitar que ações indesejadas sejam realizadas, ou seja, como o processo de execução de *applets* envolve carregamentos de arquivos de classes, isto pode acarretar, por exemplo, a execução de um arquivo de classe que formate o disco da máquina no qual está executando, causando danos irreparáveis. Logo, é por esta e outras razões de segurança que os navegadores e *applet viewers* restringem o que um *applet* pode ou não fazer.

Um problema comum com *applets*, é que cada navegador utiliza uma abordagem de implementação Java diferente, ou seja, a maioria suporta código JDK 1.02 e outros JDK 1.1. Portanto, não se pode garantir a portabilidade de um *applet* já que sua funcionalidade pode ser comprometida, dependendo do navegador ou *applet viewer* em que irá executar. Para obter mais informações sobre *applets* e suas restrições veja [8].

Partindo da aplicação *stand-alone* do TVQE construímos o *applet* TVQE, mas com algumas restrições na interface: todos os componentes gráficos utilizados para projetar a interface fazem parte do pacote AWT[12], pois a maioria dos navegadores (*Internet Explorer*, *Netscape*) utilizam este pacote como padrão de componentes gráficos; e por motivos de comodidade ao usuário, pois a versão *stand-alone* utiliza outras classes que não fazem parte deste pacote, precisando portanto de um *software*

[12] O pacote AWT é formado por um conjunto de classes divido em três categorias: os gráficos, como fontes, cores; os componentes GUI (*Graphical User Interface*), como botões, janelas; e os gerenciadores de *layouts*, que controlam os *layouts* dos componentes [7]

adicional (*java plug-in*[13]) a ser instalado pelo usuário em sua máquina local para que seu navegador reconheça e aceite estas classes.

5.4. Estrutura de Classes

Para representar a estrutura de classes que constitui o Tradutor, utilizamos o diagrama de classes que faz parte da linguagem de modelagem visual UML (*Unified Modeling Language*). UML é uma linguagem que unifica as melhores práticas criadas para modelagem de sistemas [10][26], algumas de suas características são descritas a seguir.

- UML é uma linguagem e não simplesmente uma notação para desenhar diagramas, mas uma linguagem completa para capturar o conhecimento (semântico) sobre um assunto e expressar este conhecimento (sintático);
- É usada para especificar, visualizar, construir e documentar sistemas;
- É baseada no paradigma de orientação a objetos;
- É aplicada em diferentes tipos de sistemas, domínios e métodos ou processos.

Antes de descrever o diagrama de classes do Tradutor, vamos entender o que é um diagrama de classes em UML.

O diagrama de classes em UML descreve a estrutura estática de um sistema, ou seja, como o sistema está estruturado e não como se comporta [26]. Este diagrama é composto por: **classes**, que representam entidades com características comuns, tais como atributos, operações e associações; e por **associações** que representam relacionamentos entre duas ou mais outras classes. As classes são representadas visualmente através de retângulos, cada retângulo é dividido em três partes, a primeira, contém o nome da classe, a segunda contém os atributos pertencentes a classe e a terceira contém os métodos da classe. As associações são representadas através de linhas com nomes associados, no final de cada linha tem uma expressão de multiplicidade indicando o número de relacionamentos existentes entre as instâncias das

[13] É um programa que contém arquivos JDK atualizados que fazem com que seu *applet* rode em qualquer navegador [12].

classes, esta multiplicidade pode ser, por exemplo, 0..1 (zero a 1), 1..* (um ou mais) ou * (zero ou mais). A figura 5.1 ilustra a associação entre as classes *Pessoa* e *Cidade* através do relacionamento *Mora*, os atributos e métodos das classes foram omitidos.

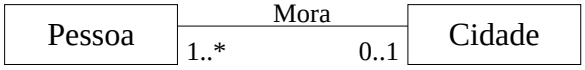


Figura 5.1 – Uma associação em UML

Existem outros tipos de associações [26], como as que representam hierarquias: **generalização**, **agregação** e **composição**. A generalização é conhecida através do relacionamento *é-um*, a agregação através do relacionamento *tem-um* e a composição pelo relacionamento *contém-um*. A agregação existe somente em situações onde a classe agregada pode ser removida sem haver a necessidade de remover as classes componentes, enquanto que na composição, a remoção da classe agregada implica na remoção das classes componentes. As agregações são representadas visualmente através de linhas com losangos brancos conectados à classe agregada, as composições através de linhas com losangos pretos conectados à classe agregada e as generalizações através de linhas com setas brancas conectadas à classe generalizada. As figuras 5.2 (a), 5.2 (b) e 5.2 (c) ilustram a representação visual de uma generalização, composição e agregação, respectivamente. Para obter maiores informações sobre UML, veja [3][10][26].

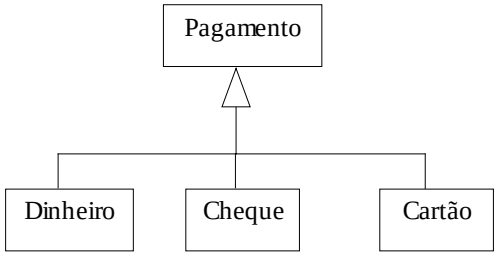


Figura 5.2 (a) – Notação da Generalização

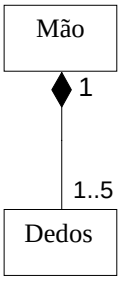


Figura 5.2 (b) – Notação de Composição

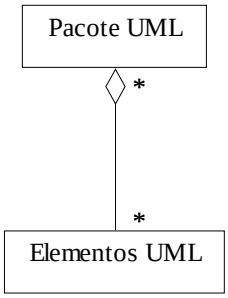


Figura 5.2 (c) – Notação de Agregação

Agora que já conhecemos um pouco sobre a linguagem UML, vamos partir para a descrição do diagrama de classes do Tradutor. As figuras 5.3, 5.4 , 5.5, 5.6, 5.7 e 5.8 ilustram os diagramas de classes em alto nível do Tradutor. Este diagrama não visualiza as variáveis de instância e os métodos de cada classe, também não foram consideradas as classes internas que fazem parte da linguagem Java. As classes que terminam com a palavra *Listener* representam as classes de eventos[14] e aquelas cujos nomes estão em itálico representam classes abstratas[15].

A classe principal tanto do Tradutor como do ambiente TVQE é a classe *agenda* ilustrada na figura 5.3. A classe *agenda* é composta pelas classes: *openListener*, *dataVisListener* e *timeVisListener*, além de várias outras responsáveis pela representação visual do ambiente TVQE, mas consideramos aqui apenas aquelas que fazem parte do Tradutor.

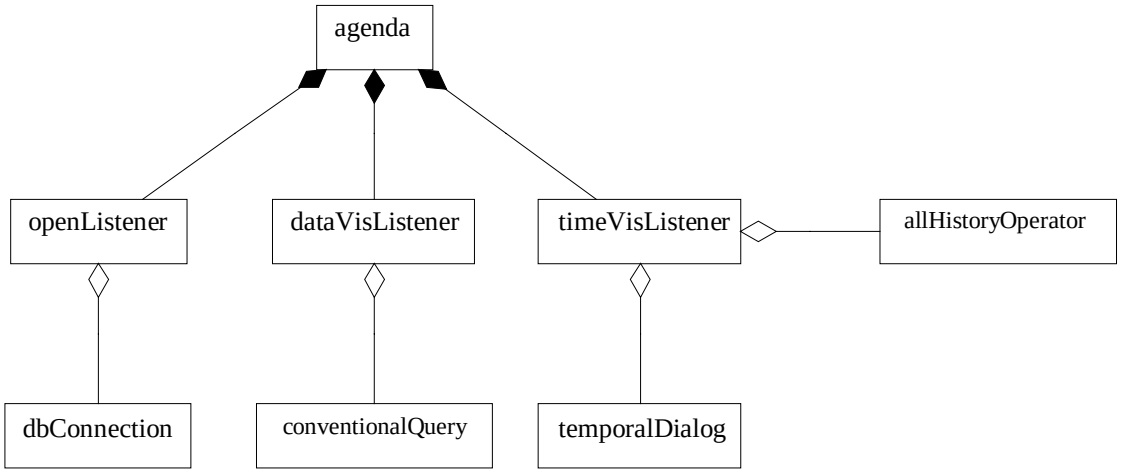


Figura 5.3 – Diagrama de Classes 1

As classes representadas na figura 5.3 são responsáveis pela execução do Tradutor.

- *openListener* – É composta entre outras pela classe *dbConnection*, que estabelece a conexão com o banco de dados. É inicializada quando o usuário ativa o botão *Open*;
- *dataVisListener* – É composta pela classe *conventionalQuery*. É inicializada quando o usuário ativa o ícone *Snapshot Vis*;

[14] São classes responsáveis pelos eventos associados aos componentes gráficos (botões, menus, etc) do TVQE.

[15] Classes abstratas são classes que não podem ser instanciadas.

- *timeVisListener* – É composta pelas classes *allHistoryOperator* e *temporalDialog*. É inicializada quando o usuário ativa o ícone *Temporal Vis*.

As classes *conventionalQuery*, *temporalDialog* e *allHistoryOpearator* serão descritas a seguir.

O diagrama da figura 5.4 ilustra as classes responsáveis pelo mapeamento para a linguagem SQL e visualização do resultado de uma consulta convencional realizada pelo usuário no TVQE.

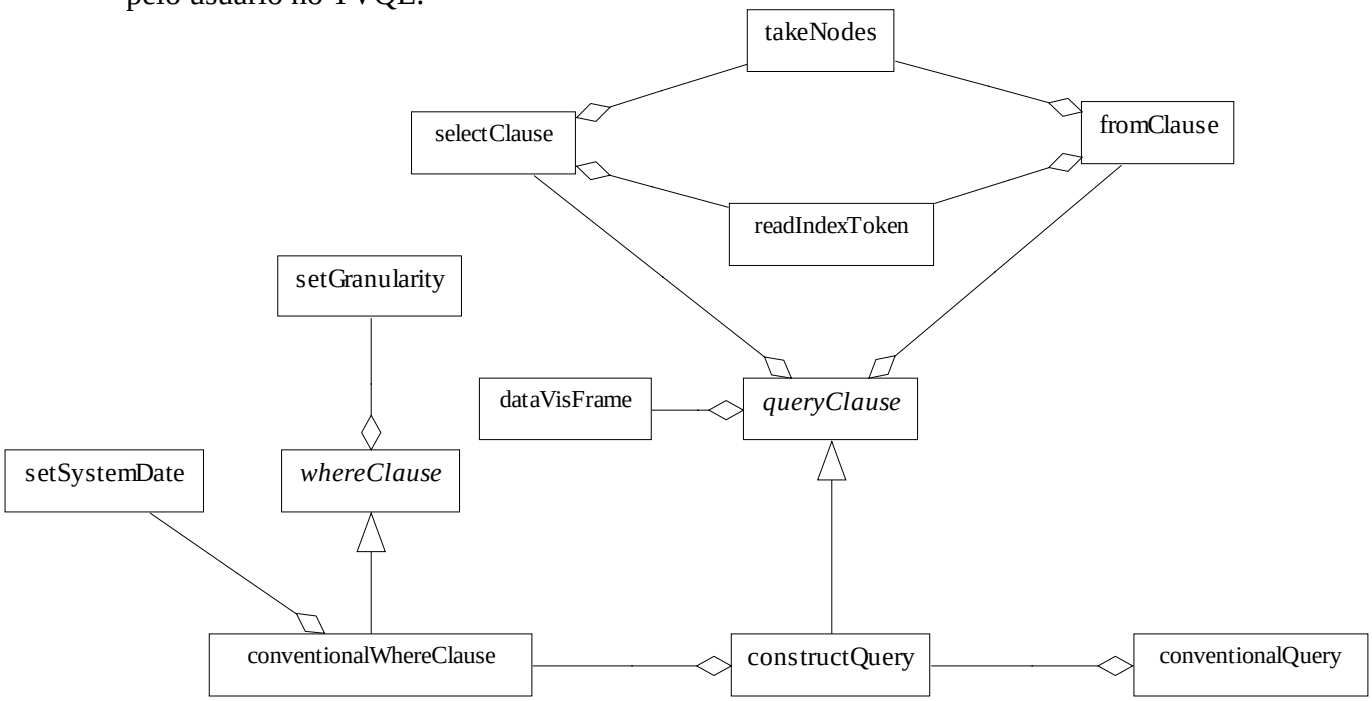


Figura 5.4 – Diagrama de Classes 2

- *conventionalQuery* – Executa a consulta convencional gerada pelo usuário visualizando seu resultado;
- *whereClause* – Gera os métodos básicos para construir qualquer condição de uma consulta realizada no ambiente TVQE;
- *setGranularity* – Transforma a granularidade do instante ou período de tempo informado pelo usuário, caso seja menor, para a dos atributos temporais das tabelas de tempo-válido referenciadas na condição da consulta;

- *queryClause* – Gera todos os métodos básicos para construir as cláusulas SELECT e FROM de uma consulta SQL e gerar a janela de visualização do resultado dessa consulta;
- *conventionalWhereClause* – Constrói a cláusula WHERE relacionada apenas às consultas convencionais realizadas no TVQE;
- *setSystemDate* – Transforma a granularidade da data do sistema para a dos atributos temporais, que fazem parte da tabela de tempo-válido referenciada na condição da consulta;
- *constructQuery* – Constrói a consulta convencional transformada em instruções SQL, através da junção das cláusulas SELECT, FROM e WHERE;
- *dataVisFrame* – Constrói a janela de visualização dos dados do resultado da consulta;
- *selectClause* - Constrói a cláusula SELECT da consulta SQL;
- *fromClause* - Constrói a cláusula FROM da consulta SQL;
- *readIndexToken* – Lê cada componente (*token*) da estrutura TGM-Relação;
- *takeNodes* - Armazena os atributos, classes e relacionamentos selecionados pelo usuário durante a consulta realizada no TVQE.

O diagrama da figura 5.5 ilustra as classes responsáveis pelo mapeamento para a linguagem SQL das consultas temporais e visualização de seus resultados.

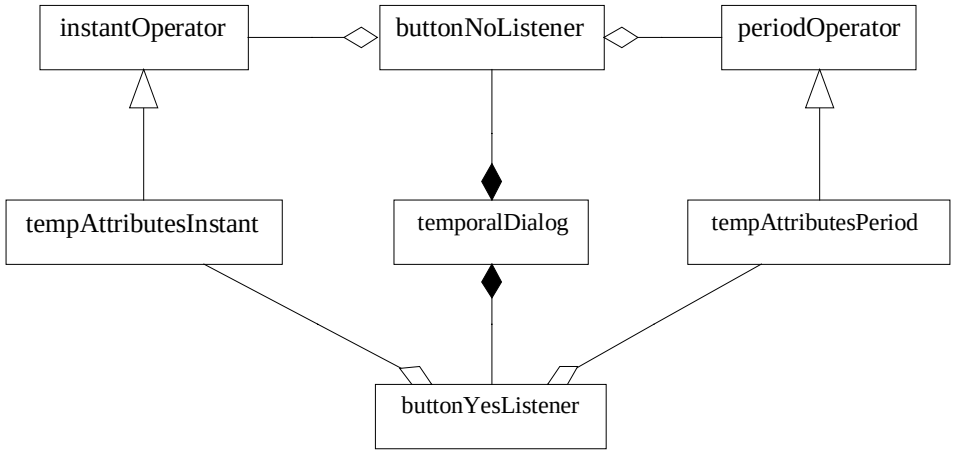


Figura 5.5 – Diagrama de Classes 3

- *ButtonYesListener* - Visualizar no resultado da consulta temporal os atributos temporais (*lifespan*) presentes na tabela de tempo-válido[16];
- *ButtonNoListener* – Não visualizar no resultado da consulta temporal os atributos temporais (*lifespan*) presentes na tabela de tempo-válido;
- As classes *tempAttributesInstant* e *tempAttributesPeriod* transformam para instruções SQL as consultas temporais realizadas pelos usuários no TVQE, cujas condições temporais utilizam os operadores de instante (*begin*, *at* e *end*) e os de período (*before, meet, during, start, finish, cross,* a*fter, contain, started-by, finished-by* e *follow*), respectivamente. Adicionando na cláusula SELECT os atributos temporais pertencentes à tabela de tempo-válido ao resultado da consulta.

O diagrama da figura 5.6 ilustra as classes responsáveis pelo mapeamento para SQL e visualização dos resultados das consultas temporais, cujas condições temporais utilizam os operadores *all history, first interval, specified interval e last interval*.

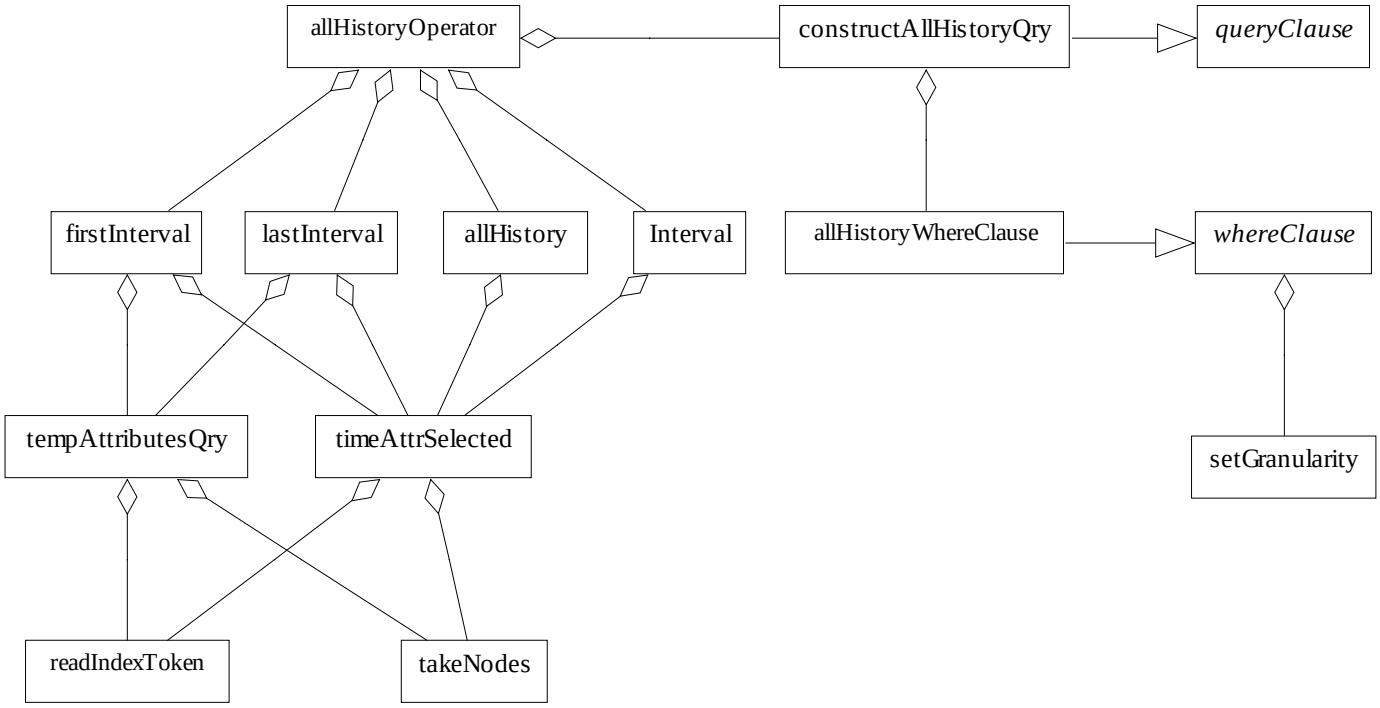


Figura 5.6 – Diagrama de Classes 4

[16] Neste caso, a tabela de tempo-válido representa uma classe, relacionamento ou atributo temporal no esquema TGM pertencente à condição da consulta temporal no TVQE.

- *AllHistoryOperator* – Executa a consulta temporal formulada pelo usuário e visualiza seu resultado, cuja condição temporal utiliza os operadores acima citados;
- *allHistoryWhereClause* – Constrói a cláusula WHERE relacionada apenas às consultas temporais, cujas condições temporais utilizam os operadores acima citados;
- As classes ***allHistory, firstInterval, Interval*** e ***lastInterval*** transformam em instruções SQL a consulta temporal, cuja condição utiliza os operadores *all history*, *first interval*, *specified interval* e *last Interval*, respectivamente;
- *constructAllHistoryQry* – Constrói as consultas temporais transformadas em instruções SQL, cujas condições utilizam os operadores acima citados, através da junção das cláusulas SELECT, FROM e WHERE;
- *tempAttributesQry* – Gera uma consulta temporal que recupera apenas o **maior** ou **menor** valor do atributo temporal **de,** pertencente a uma tabela de tempo-válido, cuja condição utiliza os operadores *last interval* ou *first interval*, respectivamente;
- *timeAttrSelected* – Recupera os nomes dos atributos temporais das tabelas de tempo-válido que irão fazer parte da cláusula SELECT da consulta temporal.

As classes *readIndexToken*, *takeNodes*, *whereClause*, *setGranularity* e *queryClause* foram descritas anteriormente, veja figura 5.4.

O diagrama da figura 5.7 ilustra as classes responsáveis pelo mapeamento para SQL e visualização dos resultados das consultas temporais, cujas condições utilizam os operadores de instante: *begin*, *at* e *end*.

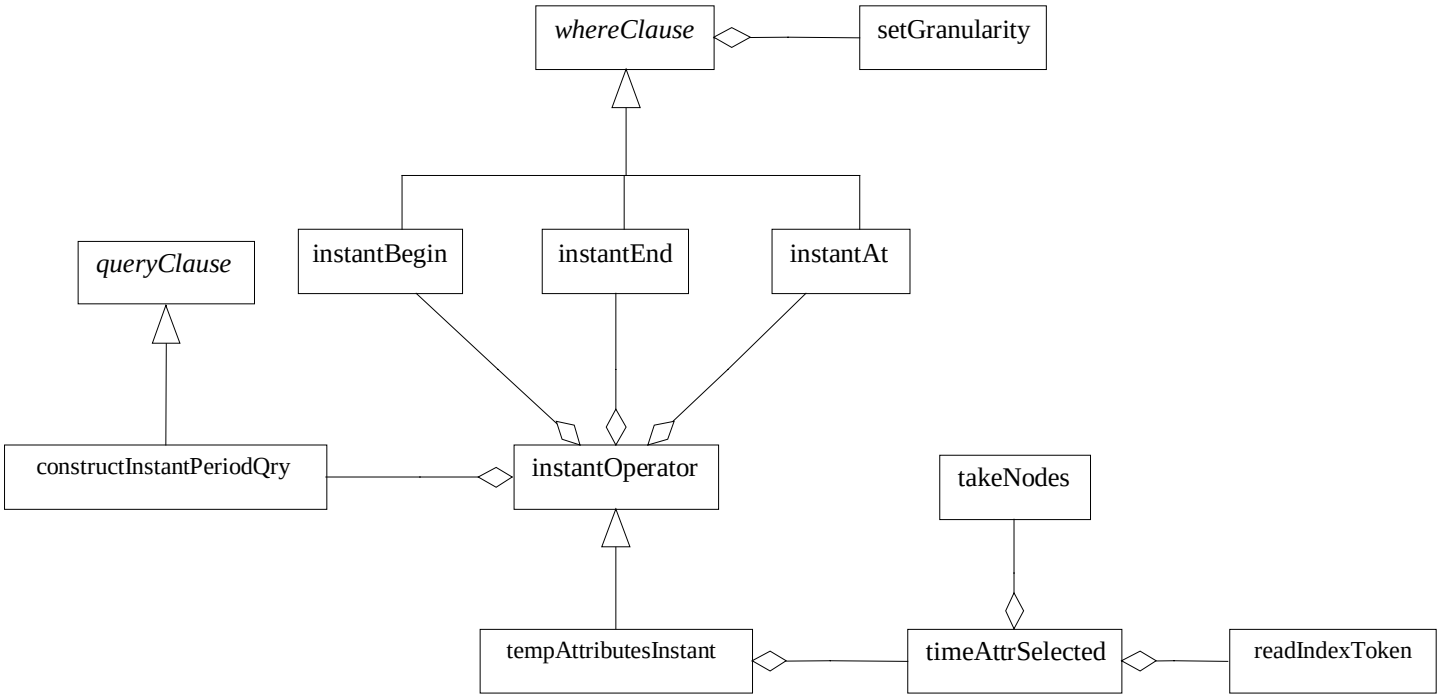


Figura 5.7 – Diagrama de classes 5

- *InstantOperator* – Executa a consulta temporal e visualiza seu resultado, cujas condições temporais utilizam os operadores de instante;
- *instantBegin* – Constrói a cláusula WHERE para o operador de instante *begin*;
- As classes *instantAt* e *instantEnd* têm a mesma funcionalidade da classe *instantBegin*, com a diferença apenas nos operadores de instante utilizado, neste caso, *at* e *end*, respectivamente;
- *constructInstantPeriodQry* – Constrói a consulta temporal transformada em instruções SQL, cuja condição utiliza os operadores de instante ou período, através da junção das cláusulas SELECT e FROM.

As classes *whereClause, queryClause, setGranularity, readIndexToken* e *takeNodes* foram descritas anteriormente na figura 5.4, *tempAttributesInstant* na figura 5.5 e *timeAttrSelected* na figura 5.6.

O diagrama da figura 5.8 ilustra as classes responsáveis pelo mapeamento para SQL e visualização dos resultados das consultas temporais, cujas condições utilizam os operadores de período: *before*, *meet*, *during*, *start*, *finish*, *cross*, a*fter*, *contain*, *started-by*, *finished-by* e *follow*.

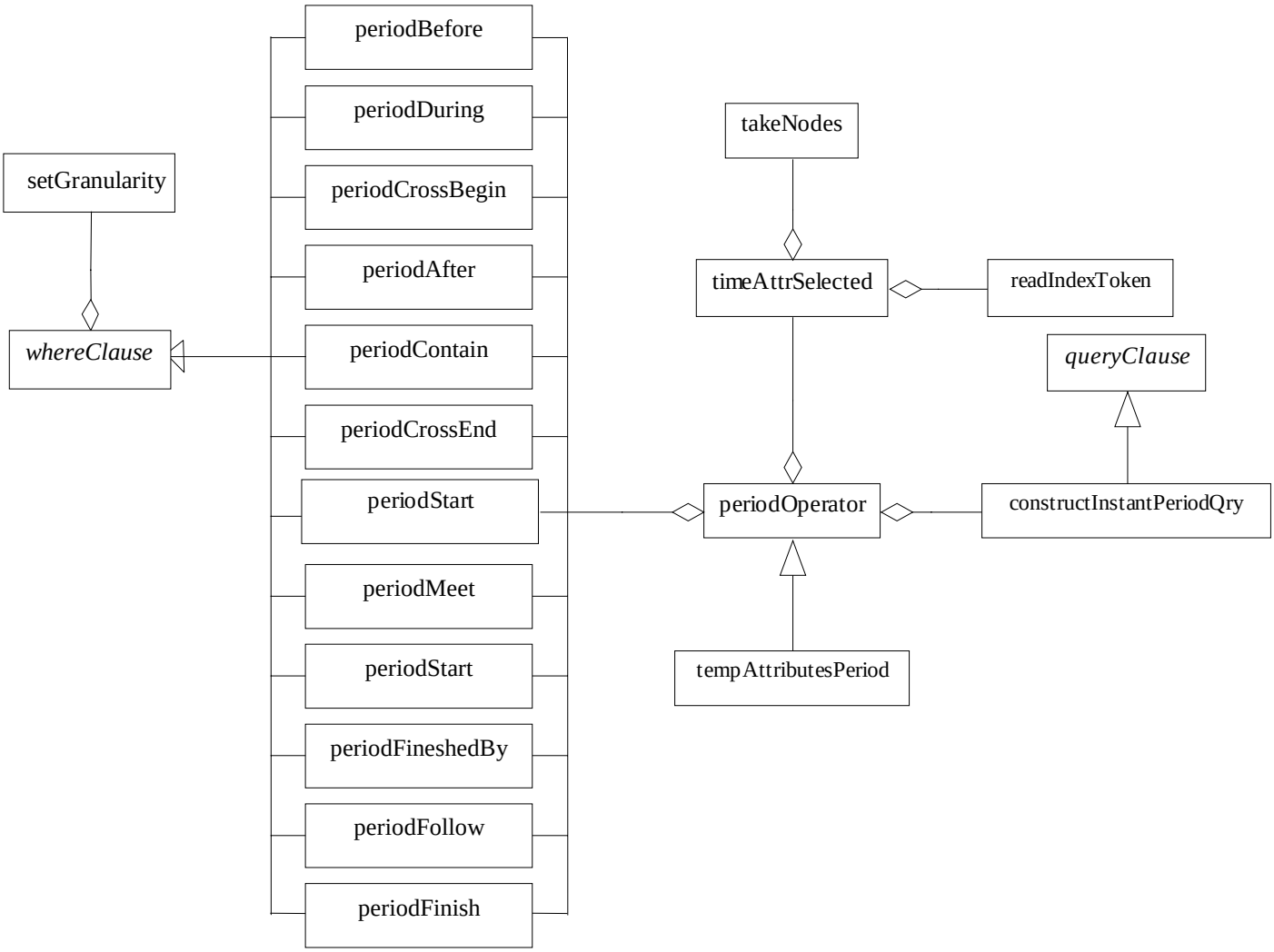


Figura 5.8 – Diagrama de Classes 6

- *periodOperator* – Executa as consultas temporais e visualiza seus resultados, cujas condições utilizam os operadores acima citados;
- *periodBefore* – Constrói a cláusula WHERE para o operador de período *before*;
- As classes ***periodMeet, periodDuring, periodStart, periodFinish, periodAfter, periodContain, periodStartedBy, periodFinishedBy*** e ***periodFollow*** têm a mesma função da classe *periodBefore*, com a mudança apenas no operador de período utilizado, que nestes casos são: *meet, during, start, finish,* a*fter, contain, started-by, finished-by* e *follow*, respectivamente; e as classes ***periodCrossBegin*** e ***periodCrossEnd*** utilizam o operador *cross*;

As classes *whereClause, queryClause, setGranularity, readIndexToken* e *takeNodes* foram descritas anteriormente na figura 5.4, *tempAttributesPeriod* na figura 5.5, *timeAttrSelected* na figura 5.6 e *constructInstantPeriodQry* na figura 5.7.

Capítulo 6 – Conclusões

Neste trabalho descrevemos as modificações e extensões realizadas no ambiente TVQE, assim como um componente, denominado Tradutor, que faz o mapeamento para a linguagem SQL das consultas realizadas no TVQE.

As extensões realizadas no TVQE tiveram o objetivo de adicionar novas funcionalidades ao ambiente, tais como: as consultas complexas, a visualização intensional do resultado da consulta e o *applet* TVQE. As modificações realizadas tiveram o objetivo de melhorar a interação do usuário com o sistema, tais como: as sincronizações dos painéis e botões, e as mensagens de ajuda ao usuário.

O Tradutor é um dos aspectos mais importantes do nosso trabalho, pois com ele é possível validar as consultas realizadas no TVQE e visualizar o resultado extensional das mesmas.

A seguir descreveremos os potenciais e limitações encontrados no trabalho, além das direções futuras de pesquisa.

6.1. Potenciais e Limitações

Podemos citar como potenciais deste trabalho:

- O ambiente TVQE permite ao usuário uma visualização intensional do resultado de uma consulta, através do esquema gráfico (*query schema*) e extensional dos dados, através da janela de visualização dos dados;
- As consultas complexas geradas permitem ao usuário a formulação de consultas mais elaboradas;
- A versão *applet* do TVQE permite que um número maior de usuários interajam com o mesmo;

- A portabilidade do Tradutor permite que as consultas transformadas em instruções SQL, possam ser executadas em qualquer SGBD;
- O Tradutor permite ao usuário visualizar tanto os dados recuperados pela consulta quanto a própria consulta escrita em SQL.

Podemos citar como limitação deste trabalho, a interface do *applet* TVQE. Esta interface tem algumas restrições relacionadas aos componentes gráficos utilizados na sua construção, devido a aspectos de implementação, ou seja, estes componente diferem visualmente dos usados na versão *stand-alone*.

6.2. Direções Futuras

Como direções futuras vamos citar os trabalhos que podem ser desenvolvidos a partir do Tradutor dentro do contexto do TVQE.

Um dos trabalhos importantes a ser feito é a criação de uma janela de visualização dos dados, utilizando o mecanismo de consulta dinâmica[17] na visualização e manipulação dos dados, pois a janela de visualização dos dados gerada no momento foi apenas para testar as consultas mapeadas para SQL, onde os dados são mostrados em uma tabela.

Outros trabalhos poderão ser realizados, tais como:

- O mapeamento das consultas complexas para a linguagem SQL;
- O melhoramento na interface das condições temporais relacionadas à referência e agregação temporal, para que as mesmas sejam transformadas em instruções SQL;
- Implementação do módulo Mapeamento TGM-Relação, que gera a estrutura TGM-Relação no formato definido no capítulo 4;

[17] A consulta dinâmica envolve um controle interativo do usuário sobre parâmetros visuais de consulta que geram um rápido e animado *display* do resultado. Para maiores informações veja [24].

- A migração do *applet* TVQE para uma arquitetura de *servlets*[18] e consequentemente a inclusão do Tradutor ao mesmo, pois devido a algumas restrições impostas pelos *browsers* não é possível fazer esta inclusão à versão original do *applet* TVQE.

[18] *Servlet* é uma classe Java que pode ser carregada dinamicamente e expandir a funcionalidade de um servidor. É muito usado com servidores de rede, onde podem substituir os *scripts* CGI. Para obter maiores informações sobre *Servlets*, veja [11].

Bibliografias e Referências

[1] Ben Shneiderman. “Designing the User Interface: Strategies for Effective Human-Computer Interaction”. Editora: Addison-Wesley, págs. 179-220, 1987.

[2] Carlo Zaniolo, Stefano Ceri, Christos Faloutsos, Richard T. Snodgrass, V. S. Subrahmanian, Roberto Zicari. “Overview of Temporal Databases”. Capítulo 5 do livro: *Advanced Database Systems.* Editora: Morgan Kaufmann, págs. 99-126, 1997.

[3] Craig Larman. “Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design”. Editora: Prentice Hall PTR, 1998.

[4] C. Jensen et al. “A Consensus Glossary of Temporal Database Concepts”. *SIGMOD RECORD*, 23(1), 1994.

[5] Carlo Batini, Stefano Ceri, Shamkant B. Navathe. “Data Modeling Concepts”. Capítulo 2 do livro: *Conceptual Database Design – An Entity-Relationship Approach*, Editora: Benjamin/Cummings, págs. 30-54, 1992.

[6] Carlo Batini, Stefano Ceri, Shamkant B. Navathe. “Logical Design for the Relational Model”. Capítulo 12 do livro: *Conceptual Database Design – An Entity-Relationship Approach*, Editora: Benjamin/Cummings, págs. 309-324, 1992.

[7] David Flanagan. “The java.awt Package”. Capítulo 18 do livro: *Java in a Nutshell: A Desktop Quick Reference*. Editora: O’reilly, págs. 279-380, 1997.

[8] David Flanagan. “Applets”. Capítulo 6 do livro: Java in a Nutshell: A Desktop Quick Reference. Editora: O’reilly, págs. 127 –143, 1997.

[9] Graham Hamilton, Rick Cattell. “JDBC™: A Java SQL API”. Publicado pela Sun Microsystems Inc., 1997.

[10] Ivar Jacobson, Grady Booch, James Rumbaugh. “The Unified Software Development Process”. Editora: Addison-Wesley, 1999.

[11] Jason Hunter, William Crawford. “Java Servlet Programming”. Editora: O’reilly, 1998.

[12] “Java™ Platform 1.2 API Specification: Solutions to Browsers Uses”. URL: http://java.sun.com/products/plugin/index.

[13] “JDBC™ Database Access from Java™: A tutorial and Annotated Reference”. Publicado pela Sun Microsystems Inc., 1997.

[14] Jim Melton, Alan R. Simon. “Understanding the New SQL: A Complete Guide”. Editora: Morgan Kaufmann, 1993.

[15] K. Vadaparty, Y. A. Aslandogan e G. Ozsoyoglu. “Towards a Unified Visual Database Access”. *ACM SIGMOD Conference, 1993, págs. 357-365*.

[16] Nina Edelweiss, J. Oliveira. “Modelagem de Aspectos Temporais de Sistemas de Informação”. IX Escola de Computação, Recife, 1994.

[17] Nina Edelweiss, Alberto H. F. Laender, João M. B. Cavalcanti, José Palazzo M. de Oliveira. “Uma abordagem para a Implementação de um Modelo Temporal Orientado a Objetos usando SGBDs Relacionais”. *X Simpósio Brasileiro de Banco de Dados*, págs. 393-407, 1995.

[18] Richard T. Snodgrass. “Language Directions”. Capítulo 12 do livro: *Developing Time-Oriented Database Applications in SQL*. Editora: Morgan Kaufmann, págs. 401-465, 2000.

[19] Shamkant B. Navathe, Rafi Ahmed. “Temporal Extensions to the Relational Model in SQL”. Capítulo 4 do livro: *Temporal Databases: Theory, Design and Implementation.* Editora: Benjamin/Cummings, págs. 92-109, 1993.

[20] S. L. Fernandes, “ConTOM – Um Sistema de Consultas Gráficas a um Banco de Dados Temporal Orientado a Objetos”. *Dissertação*, Departamento de Sistemas e Computação, UFPB - Campus II, 1995.

[21] S. L. Fernandes Silva. “Integrando Aspectos Temporais no Acesso e na Usabilidade de Interfaces Visuais Adaptativas a Banco de Dados Históricos”. Proposta para Exame de Qualificação, Departamento de Engenharia Elétrica, UFPB - Campus II, págs. 1-15, 1997.

[22] S. L. Fernandes Silva, Ulrich Schiel and T. Catarci. “Visual Query Operators for Temporal Databases”. *Proc. 4th Intl. WorkShop on Temporal Representation and Reasoning (TIME’97)*, Florida - USA, 1997.

[23] S. L. Fernandes Silva, Tiziana Catarci. “Visual Modeling of Temporal Data in Usability Experiments”. *Visual Database Systems 4 (VDB4)*, 1998.

[24] S. L. Fernandes Silva. “Formalizando Interação Visual com Banco de Dados Históricos”. *Tese de Doutorado*, Departamento de Engenharia Elétrica, UFPB - Campus II, 1999.

[25] S. L. Fernandes Silva, Ulrich Schiel. “O Modelo Temporal de Objetos TOM”. Relatório Técnico - DSC-002/96, Departamento de Sistemas e Computação, UFPB - Campus II, 1996.

[26] Sinan Si Alhir. “UML in a Nutshell: A Desktop Quick Reference”. Editora: O’Reilly & Associations, Inc., 1998.

[27] Shamkant B. Navathe, Rafi Ahmed. “Temporal Databases: A Prelude to Parametric Data”. Capítulo 2 do livro: *Temporal Databases: Theory, Design and Implementation*. Editora: Benjamin/Cummings, págs. 28-37, 1993.

[28] T. Catarci, G. Santucci, M. Angelaccio. “Fundamental Graphical Query Primitives For Visual Query Language”. *Information Systems Vol. 18, No. 2*, págs. 75-98, 1993.

[29] Ulrich Schiel. “An Abstract Introduction to The Temporal-Hierarchic Model”. *Proc. 9th. VLDB*, Florença - Itália, 1983.

[30] Ulrich Schiel. “Modelo de Dados”. Capítulo 3 da apostila: *Sistemas de Informação e Banco de Dados.* Coordenaçõao de Pós-graduação em Informática, UFPB – Campus II, 1998.

[31] Ulrich Schiel. “Sistemas de Informação Computadorizados (SIC)” . Capítulo 2 da apostila: *Sistemas de Informação e Banco de Dados.* Coordenaçõao de Pós-graduação em Informática, UFPB – Campus II, 1998.

[32] Vinícius Medina Kern. “Normalização”. Capítulo 3 do livro: *Banco de Dados Relacionais: Teoria e Prática de Projetos*. Editora: Érica, págs. 43-70, 1994.

Apêndice A - Estrutura Interna do Esquema TGM e da Árvore de Contextos

Tanto a árvore de contextos como o esquema TGM são gerados a partir da leitura de um arquivo com extensão "**.dat**". Este arquivo é a representação interna da árvore de contextos e do esquema TGM, sua estrutura é descrita a seguir.

<Número de nodos do esquema>

<Identificador do nodo, Nome do nodo, Tipo do nodo, Coordenada X do nodo, Coordenada Y do nodo>, onde **Tipo do nodo** pode ser:

0 = Contexto;

1 = Classe;

2 = Classe Temporal;

3 = Atributo;

4 = Atributo Temporal;

5 = Relacionamento;

6 = Relacionamento Temporal.

<Número de relacionamentos existentes entre os nodos do esquema>

<NodoX, NodoY, Tipo de Arco>, onde **Tipo de arco** indica o tipo de relação que pode existir entre os nodos, cada relação pode ser do tipo:

0 = Normal, associa uma classe com um atributo (temporal) ou com um relacionamento (temporal), e vice-versa;

1 = Herdado, associa uma classe com um atributo (temporal) ou um relacionamento (temporal) herdado de sua super classe;

2 = Contexto-Classe ou **Contexto-Contexto**, indica que uma classe é parte de um contexto ou que um contexto é parte de outro contexto;

3 = Superclasse-Classe, representa a relação *is-a* (é-um) entre uma classe e sua super classe;

4 = Componente-agregação, representa a relação *part-of* (parte-de) entre as classes componentes e a classe agregada.

A árvore de contextos e o esquema TGM para a agência de empregos têm suas estruturas internas descritas no arquivo *employment.dat*, ilustrado a seguir.

```
26
00 Employment_Agency 0 0.0 0.0
01 Personal_Context 0 0.0 0.0
02 Employment_Context 0 0.0 0.0
03 Person 1 0.15 0.15
04 Candidate 2 0.05 0.30
05 Employee 2 0.2 0.3
06 City 1 0.55 0.05
07 Car 1 0.55 0.3
08 Department 1 0.05 0.60
09 Company 1 0.35 0.75
10 Project 1 0.35 0.6
11 Salary 4 0.35 0.45
12 Task 3 0.2 0.6
13 Age 3 0.35 0.25
14 Name 3 0.45 0.1
15 Lives 5 0.35 0.05
16 Owns 5 0.45 0.2
17 City_Name 3 0.65 0.1
18 Plate 3 0.65 0.4
19 Dep_Name 3 0.1 0.7
20 Comp_name 3 0.55 0.7
21 Proj_name 3 0.4 0.65
22 Part_of 5 0.2 0.75
23 Job 6 0.2 0.45
24 Emp_Level 4 0.5 0.45
25 Works_in 6 0.05 0.45
```

40
00 02 2
00 01 2
01 07 2
01 06 2
01 03 2
02 10 2
02 09 2
02 08 2
02 05 2
02 04 2
03 16 0
03 15 0
03 14 0
03 13 0
03 05 3
03 04 3
04 16 1
04 15 1
04 14 1
04 13 1
05 25 0
05 24 0
05 23 0
05 16 1
05 15 1
05 14 1
05 13 1
05 11 0
06 17 0
07 18 0
08 22 0
08 19 0

09 20 0

10 21 0

15 06 0

16 07 0

22 09 0

23 10 0

23 12 0

25 08 0

Apêndice B – *Script* de Criação das Tabelas

```
/* Table: CANDIDATO */
CREATE TABLE CANDIDATO (CANDIDATOID INTEGER NOT NULL,
      DE VARCHAR(10) NOT NULL,
      ATE VARCHAR(10) NOT NULL,
PRIMARY KEY (CANDIDATOID, DE, ATE));

/* Table: CARRO */
CREATE TABLE CARRO (CARROID INTEGER NOT NULL,
      PLACA VARCHAR(10),
      PESSOAID INTEGER NOT NULL,
PRIMARY KEY (CARROID));

/* Table: CIDADE */
CREATE TABLE CIDADE (CIDADEID INTEGER NOT NULL,
      NOME_CID VARCHAR(50),
PRIMARY KEY (CIDADEID));

/* Table: COMPANHIA */
CREATE TABLE COMPANHIA (COMPID INTEGER NOT NULL,
      NOME_COMP VARCHAR(60),
PRIMARY KEY (COMPID));
```

```
/* Table: DEPARTAMENTO */
CREATE TABLE DEPARTAMENTO (DEPARTAMENTOID INTEGER NOT NULL,
    NOME_DEP VARCHAR(60),
PRIMARY KEY (DEPARTAMENTOID));

/* Table: EMPREGADO */
CREATE TABLE EMPREGADO (EMP_ID INTEGER NOT NULL,
    DE VARCHAR(10) NOT NULL,
    ATE VARCHAR(10) NOT NULL,
PRIMARY KEY (EMP_ID, DE, ATE));

/* Table: EMPREGADO_NIVEL */
CREATE TABLE EMPREGADO_NIVEL (EMP_ID INTEGER NOT NULL,
    EMP_NIVEL INTEGER,
    DE VARCHAR(10) NOT NULL,
    ATE VARCHAR(10) NOT NULL,
PRIMARY KEY (EMP_ID, DE, ATE));

/* Table: EMPREGADO_SALARIO */
CREATE TABLE EMPREGADO_SALARIOS (EMP_ID INTEGER NOT NULL,
    SALARIO FLOAT,
    DE VARCHAR(10) NOT NULL,
    ATE VARCHAR(10) NOT NULL,
PRIMARY KEY (EMP_ID, DE, ATE));

/* Table: TRABALHA */
CREATE TABLE TRABALHA (EMP_ID INTEGER NOT NULL,
    PROJETOID INTEGER NOT NULL,
    DE VARCHAR(10) NOT NULL,
    ATE VARCHAR(10) NOT NULL,
    ATIVIDADE VARCHAR(100),
PRIMARY KEY (EMP_ID, PROJETOID, DE, ATE));
```

```
/* Table: PESSOA */
CREATE TABLE PESSOA (PESSOAID INTEGER NOT NULL,
    IDADE INTEGER,
    NOME VARCHAR(45),
    CIDADEID INTEGER NOT NULL,
PRIMARY KEY (PESSOAID));

/* Table: PROJETO */
CREATE TABLE PROJETO (PROJETOID INTEGER NOT NULL,
    NOME_PROJ VARCHAR(100),
PRIMARY KEY (PROJETOID));

/* Table: TRABALHA_EM */
CREATE TABLE TRABALHA_EM (EMP_ID INTEGER NOT NULL,
    DEPARTAMENTOID INTEGER NOT NULL,
    DE VARCHAR(10) NOT NULL,
    ATE VARCHAR(10) NOT NULL,
PRIMARY KEY (EMP_ID, DEPARTAMENTOID, DE, ATE));

/* REFERENTIAL INTEGRITY */
ALTER TABLE PESSOA ADD FOREIGN KEY (CIDADEID) REFERENCES CIDADE(CIDADEID);
ALTER TABLE EMPREGADO_SALARIO ADD FOREIGN KEY (EMP_ID) REFERENCES PESSOA(PESSOAID);
ALTER TABLE TRABALHA ADD FOREIGN KEY (EMP_ID) REFERENCES PESSOA(PESSOAID);
ALTER TABLE TRABALHA ADD FOREIGN KEY (PROJETOID) REFERENCES PROJETO(PROJETOID);
ALTER TABLE CARRO ADD FOREIGN KEY (PESSOAID) REFERENCES PESSOA(PESSOAID);
ALTER TABLE CANDIDATO ADD FOREIGN KEY (CANDIDATOID) REFERENCES PESSOA(PESSOAID);
ALTER TABLE EMPREGADO ADD FOREIGN KEY (EMP_ID) REFERENCES PESSOA(PESSOAID);
ALTER TABLE COMPANHIA ADD FOREIGN KEY (COMPID) REFERENCES DEPARTAMENTO(DEPARTAMENTOID);
```

```
ALTER TABLE EMPREGADO_NIVEL ADD FOREIGN KEY (EMP_ID) REFERENCES PESSOA(PESSOAID);
ALTER TABLE TRABALHA_EM ADD FOREIGN KEY (EMP_ID) REFERENCES PESSOA(PESSOAID);
ALTER TABLE TRABALHA_EM ADD FOREIGN KEY (DEPARTAMENTOID) REFERENCES DEPARTAMENTO(DEPARTAMENTOID);
```