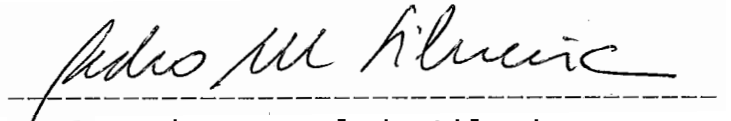
UM GERADOR DE TRANSAÇÕES VOLTADO PARA O MODELO ENTIDADE-RELACIONAMENTO

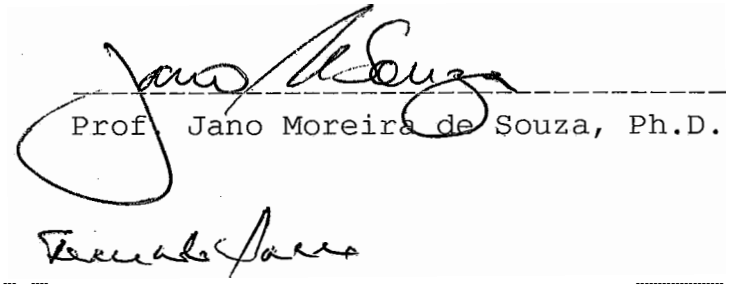
CLÁUDIO DEGRAZIA RIBEIRO

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:



Prof. Pedro Manoel da Silveira, Ph.D.
(Presidente)



Prof. Jano Moreira de Souza, Ph.D.

Prof. Fernando Silva Pereira Manso, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

MAIO DE 1992

RIBEIRO, CLÁUDIO DEGRAZIA

Um Gerador de Transações Voltado Para o Modelo Entidade-Relacionamento [Rio de Janeiro], 1992

viii, 110p., 29,7cm. (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1992)

Tese – Universidade Federal do Rio de Janeiro

1. Reutilização de Software 2. Geradores de Aplicação 3. Modelo Entidade-Relacionamento 4. Bancos de Dados

I. COPPE/UFRJ II. Título(série).

A Fátima.

AGRADECIMENTOS

A minha esposa Fátima, por estar sempre ao meu lado, em todos momentos da minha vida.

Aos meus pais, Paulo e Ivone, que proporcionaram as bases da minha educação, sempre me incentivando a atingir grandes metas.

A minha irmã, Nádia, pelo carinho e amizade sempre proporcionados.

Ao professor e orientador Pedro Manoel, pelo incentivo, pela dedicação e pela valiosa orientação recebidos.

Ao professor e co-orientador Jano, pelas inúmeras sugestões e pela colaboração em minha formação acadêmica.

Ao professor Fernando, pelo tempo dedicado à observação deste trabalho.

A equipe do UniversiData em geral, e em especial ao Delvaux, Helena e Pinhel, pelo apoio e inúmeras sugestões.

Ao Núcleo de Computação Eletrônica da UFRJ, pela minha formação humana e profissional e pela possibilidade de desenvolvimento desta tese.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos para a obtenção do grau de Mestre em Ciências (M.Sc.).

UM GERADOR DE TRANSAÇÕES VOLTADO PARA O MODELO ENTIDADE-RELACIONAMENTO

Cláudio Degrazia Ribeiro

Maio de 1992

Orientador: Pedro Manoel da Silveira

Programa : Engenharia de Sistemas e Computação

O objetivo desta dissertação consiste na definição e implementação de um gerador de aplicações que tem como base uma extensão do Modelo Entidade-Relacionamento. Este gerador de aplicações, denominado Gerador de Transações, recebe como entrada a descrição do esquema E-R especificado e gera automaticamente um conjunto de telas e transações para a manutenção da base de dados de uma aplicação.

Em seu atual estágio, o Gerador de Transações é capaz de gerar programas para os tipos de transação mais usuais como inclusão, exclusão e alteração de entidades e relacionamentos. Nestas operações, são respeitadas as propriedades do esquema E-R tais como cardinalidades, dependências de inclusão e exclusão e chaves. O Gerador de Transações utiliza a biblioteca de métodos do TURBO PASCAL 6.0 para a definição de interfaces extremamente elaboradas. As aplicações geradas possuem tanto padrões repetitivos para quaisquer transações de atualização geradas, quanto partes variáveis que vão depender do esquema E-R definido pelo usuário.

O gerador de aplicações proposto é prático, aberto a extensões e integra conhecimentos de modelagem de dados, restrições de integridade e geração de aplicações. Tudo isso inserido num ambiente completo de projeto e manipulação de bancos de dados construídos com base no Modelo Entidade-Relacionamento.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

A TRANSACTIONS GENERATOR BASED UPON THE ENTITY-RELATIONSHIP MODEL

Cláudio Degrazia Ribeiro

May, 1992

Thesis Supervisor: Pedro Manoel da Silveira

Department: Systems Engineering and Computing

The main purpose of this dissertation is the specification and implementation of an application generator based upon an extension of the Entity-Relationship Model. This application generator, called Transactions Generator, accepts the description of an E-R schema as input and automatically generates a set of screens and transactions for the maintenance of the database of an application.

The Transactions Generator can generate programs that perform usual kinds of transactions: instance inclusion, instance exclusion and attribute update for entities and relationships. Those transactions obey the integrity constraints of the database scheme, such as cardinalities, inclusion and exclusion dependencies and keys. A TURBO PASCAL 6.0 methods library is used for the definition of elaborated interfaces. The generated applications have repeated patterns for all database update transactions and also have variant parts which depend on the specified E-R schema.

The proposed application generator is practical, easy to extend and integrates topics such as data modelling, integrity constraint and application generation. It is also part of a complete environment for project and manipulation of databases built under an Entity-Relationship like data model.

ÍNDICE

Capítulo I _ INTRODUÇÃO **1**

Capítulo 11 _ REUTILIZAÇÃO DE SOFTWARE **4**
11.1 _ ALGUMAS CONSIDERAÇÕES 4
II.1.1 _ DILEMAS DA REUTILIZAÇÃO 5
II.1.2 _ O PROCESSO DE REUTILIZAÇÃO 6
II.1.3 _ FATORES ECONÔMICOS 7
II.2 - CLASSIFICAÇÃO DAS TECNOLOGIAS DE REUTILIZAÇÃO 7
II.2.1 - TECNOLOGIAS DE COMPOSIÇÃO 8
II.2.2 - TECNOLOGIAS DE GERAÇÃO 8
II.2.2.1 - SISTEMAS BASEADOS EM LINGUAGEM 9
II.2.2.2 - SISTEMAS BASEADOS EM TRANSFORMAÇÃO 9
II.2.2.3 - SISTEMAS BASEADOS EM DEDUÇÃO 10
II.2.2.4 - GERADORES DE APLICAÇÃO 10
11.3 - A CONSTRUÇÃO DE GERADORES DE APLICAÇÃO 11
II.3.1 - A ESTRUTURA DE UM GERADOR DE APLICAÇÃO 12
II.3.2 - UMA ABORDAGEM PARA A CONSTRUÇÃO DE GERADORES 13
II.3.3 - COMPONENTES BÁSICOS DOS GERADORES DE APLICAÇÃO .. 16
II.4 - PERSPECTIVAS DA PROGRAMAÇÃO AUTOMÁTICA 17

capítulo III _ MODELOS DE DADOS **19**
III.1 _ INTRODUÇÃO 19
111.2 _ QUALIDADE DE MODELOS E ESQUEMAS CONCEITUAIS 20
III.3 _ O MODELO ENTIDADE-RELACIONAMENTO 21
III.3.1 _ INTRODUÇÃO 21
III.3.2 _ COMPONENTES BÁSICOS 22
III.3.3 _ EXTENSÕES 24
111.4 _ RESTRIÇÕES DE INTEGRIDADE 25
III.4.1 _ RESTRIÇÕES 25
III.4.2 _ RESTRIÇÕES DE INTEGRIDADE NO MODELO RELACIONAL . 26
III.4.3 _ RESTRIÇÕES DE CARDINALIDADE NO MODELO E-R 27

Capítulo IV _ O PROJETO UNIVERSIDATA 29
IV.1 _ ORIGENS E PROPOSTAS 29
IV.2 _ FACILIDADES DO QUICK-DB 30
IV.3 _ OS COMPONENTES DO QUICK-DB 31
IV.4 _ CONCLUSÃO 34

Capítulo V _ A SOLUÇÃO PROPOSTA **35**
V.1 _ INTRODUÇÃO 35
V.2 _ DEFINIÇÃO DAS CARACTERÍSTICAS 35
V.3 _ RESTRIÇÕES DE INTEGRIDADE NAS TRANSAÇÕES GERADAS 36
V.3.1 _ INTEGRIDADE DE ENTIDADE 37
V.3.2 - UNICIDADE DE CHAVE 37
V.3.3 - INTEGRIDADE REFERENCIAL 38
V.3.4 - RESTRIÇÕES DE CARDINALIDADE RELATIVAS 38
V.4 - CONCEITO DE MACROTRANSAÇÃO E MICROTRANSAÇÃO 41
V.5 - A CONSTRUÇÃO DO GERADOR DE TRANSAÇÕES 42
V.5.1 - ARQUITETURA DO GERADOR DE TRANSAÇÕES 42

V.5.2 - ABORDAGEM DE CONSTRUÇÃO DO GERADOR DE TRANSAÇÕES . 43
V.5.3 - COMPONENTES BÁSICOS DO GERADOR DE TRANSAÇÕES 44
V.6 - ESPECIFICAÇÃO DAS TRANSAÇÕES DE ATUALIZAÇÃO 44
V.6.1 - ALTERAÇÃO DE ENTIDADE 45
V.6.2 - ALTERAÇÃO DE RELACIONAMENTO 45
V.6.3 - EXCLUSÃO DE ENTIDADE 46
V.6.4 - EXCLUSÃO DE RELACIONAMENTO 47
V.6.5 - INCLUSÃO DE ENTIDADE 48
V.6.6 - INCLUSÃO DE RELACIONAMENTO 50

capítulo VI _ IMPLEMENTAÇÃO 51
VI.l _ ASPECTOS DE ORIENTAÇÃO A OBJETOS 51
VI.2 _ BANCO DE DADOS AUTÔNOMO 52
VI.2.1 _ SERVIDOR E-R 52
VI.2.2 _ OBJETOS GERADOS 54
VI.2.3 _ MÉTODOS DE ACESSO 54
VI.3 _ **INTERFACE** COM O USUÁRIO 56
VI.4 _ ESTRUTURA DO GERADOR DE TRANSAÇÕES 57
VI.5 _ CONSISTÊNCIAS E ERROS DAS TRANSAÇÕES 59
VI.6 _ ABRANGÊNCIA 60

Capítulo VII _ CONCLUSÕES 63
VII.l _ CONTRIBUIÇÕES DO PROJETO 63
VII.2 _ CONTRIBUIÇÕES DO GERADOR DE TRANSAÇÕES 63
VII.3 _ CONTINUIDADE DO TRABALHO 64
VII.3.1 _ GERADOR DE RELATÓRIOS 64
VII.3.2 _ MÓDULO DE CONSULTAS 65
VII.3.3 _ INTERFACE WINDOWS 65
VII.3.4 _ REUTILIZAÇÃO DAS TRANSAÇÕES POR PROGRAMAS 65
VII.3.5 - CONSISTÊNCIA DE ATRIBUTOS 66
VII.3.6 - MECANISMO DE **ESCAPE** 66
VII.3.7 - ESPECIFICAÇÃO FUNCIONAL 66
VII.4 - ANÁLISE COMPARATIVA 67

REFERÊNCIAS BIBLIOGRÁFICAS 69

APÊNDICE _ EXEMPLOS DE UTILIZAÇÃO 74

CAPÍTULO 1 – INTRODUÇÃO

A sempre crescente demanda por sistemas de **software** cada vez mais abrangentes e complexos e a ausência de uma forma mais organizada e sistematizada no processo de seu desenvolvimento resultaram na chamada "crise de **software**". Ainda que tenham surgido novas linguagens de programação, tenham sido estabelecidos novos conceitos, técnicas e metodologias e tenham sido criadas ferramentas ou mesmo ambientes integrados para o desenvolvimento de **software;** nota-se claramente um grande descompasso entre as evoluções do **hardware** e do **software** até os dias de hoje [BALZ 1985].

A situação em 1984 segundo [HORO 1984] é a seguinte: A produtividade na criação de **software** tem aumentado a uma taxa de 3% a 8% ao ano nos últimos 20 anos, enquanto a capacidade de processamento instalada tem subido a uma taxa de mais de 40% ao ano. Há evidências de que este quadro ainda é válido para 1992.

Pode-se citar alguns desafios que o desenvolvimento de **software** vem enfrentando [HORO 1985]: demanda crescente por novas aplicações, complexidade crescente das novas aplicações, custos crescentes e pouca disponibilidade de pessoas habilitadas. É extremamente improvável que estes desafios possam ser enfrentados com a utilização de metodologias e ferramentas de engenharia de **software** convencionais.

Neste contexto, surge um novo conceito, com o objetivo de aumentar a produtividade de **software** em uma ordem de grandeza ou mais e com isso diminuir muito os custos de desenvolvimento. A esse conceito, deu-se o nome de reutilização de **software.**

Segundo [JONE 1984], menos do que 15% do código escrito é único, novo e específico para cada aplicação. Os restantes 85% parecem ser comuns, genéricos para mais de uma aplicação. Mesmo assim, geralmente se inicia a construção do **software** como se ele fosse único e singular, não se aproveitando dos esforços já dispendidos em outros projetos.

Para que se obtenha um nível de reutilização viável, são necessários alguns passos prévios, tais como : identificar as funções específicas que ocorrem com mais freqüência, encapsular estas funções, prover ferramentas e suporte para tornar a reutilização fácil e definir padrões. Esses passos fazem com que se estabeleça um mecanismo de transmissão do conhecimento sobre o produto para que depois ele possa ser efetivamente reutilizado. A reutilização também possibilita a redução dos custos de manutenção, pois os componentes tendem a ser largamente testados nos sistemas em que fazem parte [FARI 1991].

A reutilização e a automação não devem ser encaradas como alternativas diferentes para aumentar a produtividade e sim como técnicas complementares. A automação tenta levar o máximo possível de trabalho para o computador, ao passo que a reutilização tenta fazer com que o uso de atividades que não podem ser automatizadas seja o mais eficiente possível [BARN 19911.

Uma das categorias em que se dividem as tecnologias de reutilização consiste nos geradores de aplicação. Um gerador de aplicação é um software que é projetado para ajudar usuários-finais a construir suas aplicações em um determinado domínio. O software possui um conhecimento muito específico sobre o domínio para o qual a aplicação vai ser gerada e isto permite que sua linguagem de especificação seja muito precisa [HORO 1984].

Esta tese apresenta algumas questões relacionadas ao conceito de reutilização de software, enfatizando a abordagem de reutilização por padrões ou reutilização por geração. Descreve uma proposta de um gerador de aplicações simples para manutenção de bases de dados a partir da descrição do domínio da aplicação contida no esquema E-R. Esta geração se beneficia do ambiente oferecido pelo QUICK-DB.

O segundo capítulo compreende um estudo sobre reutilização de software. São analisadas as principais classes em que podemos dividir a programação automática. Faz uma descrição mais pormenorizada dos geradores de aplicação. São feitas ainda algumas referências a outros trabalhos sobre reutilização a fim de consolidar os conceitos apresentados.

O terceiro capítulo engloba um estudo sobre modelagem de dados, principalmente no que concerne o modelo E-R. São apresentadas as principais características desse modelo, no qual está baseado este trabalho; e são abordadas algumas propostas de extensão para o mesmo. Por fim, apresenta um estudo sobre restrições de integridade, que será utilizado para que as aplicações geradas mantenham sempre o banco de dados íntegro.

No capítulo IV, será descrito o projeto no qual encontra-se inserido o presente trabalho. Será apresentado o ambiente QUICK-DB, que integra todos os módulos implementados.

O quinto capítulo descreve a ferramenta proposta neste trabalho, denominada Gerador de Transações, através de seus fundamentos, características e da especificação de um protótipo. É mostrada ainda a importância das restrições de integridade para o sucesso das transações geradas e é feito um estudo sobre a construção do Gerador de Transações baseado em alguns conceitos teóricos.

O sexto capítulo apresenta a proposta de implementação baseada na especificação do capítulo anterior. Descreve os métodos de acesso ao banco de dados e como as transações geradas utilizam estes métodos. Faz uma proposta de interface com o usuário, descreve a estrutura do programa gerador e mostra as consistências verificadas na realização de uma transação.

Finalmente, o sétimo capítulo descreve os principais resultados obtidos neste trabalho como um todo. São destacadas as perspectivas de continuidade e é feita uma análise comparativa em relação a outros trabalhos.

No apêndice, é apresentado um exemplo de utilização desde o esquema E-R, a partir do qual se originou a aplicação, até a execução de algumas transações geradas.

CAPÍTULO II – REUTILIZAÇÃO DE **SOFTWARE**

"Por que o software não é como o hardware? Por que todo novo desenvolvimento deve começar sempre do mesmo ponto? Deveria haver catálogos de módulos de software, assim como existem catálogos de circuitos VLSI: Sempre que construímos um novo sistema, deveríamos escolher componentes destes catálogos e combiná-los, ao invés de reinventar a roda a cada vez. Nós escreveríamos menos software e talvez fizéssemos um trabalho melhor naquilo que estamos desenvolvendo. Com isso, os problemas que todos lamentam - os altos custos, prazos ultrapassados, falta de confiabilidade - simplesmente não desapareceriam? Por que não é assim?" [MEYE 1987].

II.1 – ALGUMAS CONSIDERAÇÕES

Antes de começar a discutir alguns aspectos da reutilização, é importante definir o conceito de domínio de aplicação. Pode-se defini-lo como sendo uma área de aplicação, ou esfera de atividades, para a qual são desenvolvidos os sistemas de programação [FARI 1991]. Na análise de domínio, são identificados objetivos e operações comuns a todos os sistemas dentro do mesmo domínio. Os componentes resultantes dessa análise são mais apropriados para o processo de reutilização porque eles capturam a funcionalidade essencial requerida no domínio de aplicação.

O conceito de reutilização tem sido largamente difundido como uma maneira de melhorar a qualidade e a produtividade no desenvolvimento de software. Em 1967, foi proposta a idéia de um catálogo de componentes de software, da mesma maneira como é feito com componentes mecânicos e eletrônicos. A aplicação desta idéia foi demonstrada para um domínio limitado com excelentes resultados. As indústrias de software japonesas também obtiveram grandes melhoramentos na produtividade com a reutilização através da integração de técnicas de diferentes disciplinas, tais como gerência de recursos, engenharia de produção, controle de qualidade, engenharia de software e psicologia industrial [DIAZ 19871.

Embora a reutilização seja uma estratégia muito promissora, estas promessas não têm se realizado e ainda permanecem sendo um desafio. Quais são as razões que impedem a concretização dessas promessas? Algumas dessas razões serão discutidas a seguir:

II.1.1 - DILEMAS DA REUTILIZAÇÃO

[BIGG 1987] apresenta vários dilemas da reutilização. A idéia geral do dilema é que uma mudança positiva em um parâmetro leva a uma mudança negativa em um outro.

. **Generalidade X Benefício** : Tecnologias que são muito gerais tem um benefício muito menor do que aquelas que focalizam apenas um ou dois domínios de aplicação. Por exemplo, um gerador de aplicação que gera telas e a **interface** com o banco de dados traz mais benefícios para o usuário do que uma linguagem de alto nível, mas suas possibilidades de uso são bem mais restritas. Na figura 1 [BIGG 1987], encontra-se uma caracterização de diversas tecnologias de reutilização segundo esse dilema.

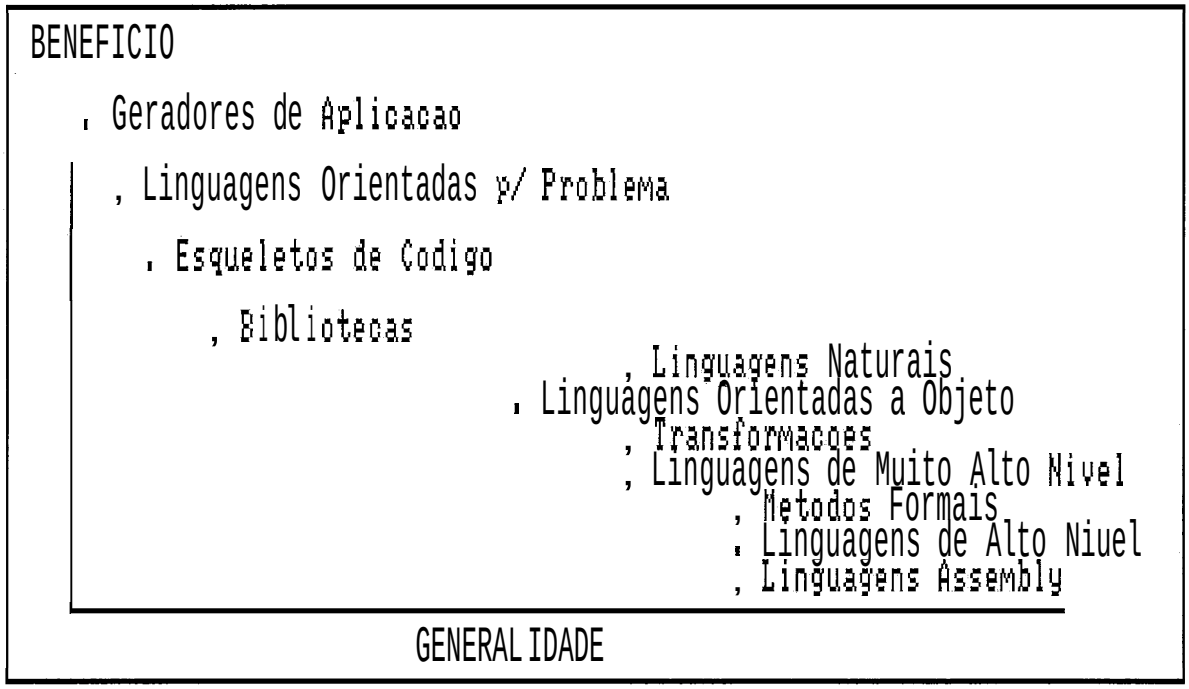


FIGURA 1 - CARACTERIZACAO DE TECNOLOGIAS DE REUTILIZACAO

. **Tamanho do Componente X Potencial de Reutilização** : Quanto maior um componente, maior o benefício com a sua reutilização. Entretanto, quanto maior o componente, mais específico ele se torna, diminuindo suas chances de uso e aumentando o custo de reutilização por causa das modificações necessárias.

. **Custo da Biblioteca de Reutilização** : É preciso um grande investimento inicial na criação de uma biblioteca rica em componentes reutilizáveis. O benefício desse investimento ocorre a longo prazo e por isso a maior parte das organizações adiam esse investimento inicial apesar do grande potencial de retorno.

II.1.2 – O PROCESSO DE REUTILIZAÇÃO

O processo de reutilização envolve alguns passos para que seja eficaz [DIAZ 1987] [BIGG 19871:

. **Classificação** : **A** classificação é uma atividade primordial para que a reutilização seja uma abordagem importante no desenvolvimento de **software.** Uma estrutura de classificação bem definida é essencial para o projeto de um sistema de recuperação eficiente. O esquema de classificação e o sistema de recuperação devem ajudar seus usuários a identificar componentes similares. Um exemplo de sistema de classificação é encontrado em [DIAZ 19871.

. **Acesso :** A partir de determinados requisitos, busca-se encontrar componentes iguais ou similares aos desejados. Um componente similar ao ideal que precisa ser parcialmente redesenvolvido reduz esforços e elimina defeitos em relação ao desenvolvimento de componentes inteiramente novos.

. **Entendimento** : O processo de entendimento é necessário para que se saiba se alguma modificação é necessária e quais esforços serão necessários para a realização desta modificação. O entendimento vai depender de fatores, tais como experiência do usuário e do tamanho, complexidade, documentação e linguagem do componente de **software.**

. **Modificação :** O processo de modificação pode ser considerado o cerne da reutilização. Ele muda a percepção de uma biblioteca estática de blocos pré-definidos para um sistema dinâmico de componentes que originam novos componentes a partir dos requisitos do novo domínio. As dificuldades de modificação vão depender das diferenças existentes entre os requisitos e os componentes existentes.

. **Composição :** Consiste na integração do componente encontrado, modificado ou não, ao **software** em desenvolvimento.

II.1.3 - FATORES ECONÔMICOS

A reutilização é um processo vital no processo de desenvolvimento de software e os investimentos feitos para sua utilização eficaz devem trazer um retorno positivo na relação custo-benefício.

Segundo [BARN 1991], os custos nessa relação representam todos os investimentos feitos para aumentar a reutilização. Eles não incluem o trabalho de desenvolver as atividades primárias do sistema, mas incluem todo o trabalho necessário para tornar estas atividades passíveis de futura reutilização. Por exemplo, as horas de trabalho destinadas especificamente a classificar e colocar componentes em uma biblioteca de reutilização, visto que isto é feito com o objetivo de beneficiar atividades subseqüentes.

Os benefícios dessa relação representam todos os ganhos advindos dos investimentos de reutilização prévios. Para calcular esse benefício, é preciso primeiro estimar o custo da atividade sem reutilização e comparar com o seu custo com reutilização.

A relação benefício-custo deve ser a maior possível. Se essa relação for menor do que 1, o investimento não compensou financeiramente. Quanto maior a proporção, maior o retorno para os investimentos feitos em reutilização. [BARN 1991] detalha ainda como maximizar essa proporção através de três estratégias: aumentar o nível de reutilização, reduzir os custos de reutilização e reduzir os investimentos feitos para que a reutilização traga benefícios.

11.2 - CLASSIFICAÇÃO DAS TECNOLOGIAS DE REUTILIZAÇÃO

A questão da reutilização pode ser abordada sob diferentes pontos-de-vista. [BIGG 1984] faz um estudo da questão sob o ponto-de-vista da tecnologia. Como as diferentes tecnologias trabalham e em que elas diferem? Dependendo da natureza dos componentes envolvidos, pode-se dividir as diferentes abordagens tecnológicas em dois grandes grupos: tecnologias de composição e tecnologias de geração.

II.2.1 – TECNOLOGIAS DE COMPOSIÇÃO

Este grupo caracteriza-se pelo fato de seus componentes serem atômicos e idealmente passivos. Os componentes são atômicos no sentido de que mantêm suas identidades individuais ao longo de sua utilização. O ideal de passividade nem sempre é alcançado e os componentes podem sofrer algum tipo de alteração para se adequar às finalidades computacionais de uma reutilização melhor. Exemplos desses componentes são esqueletos de código, subrotinas, funções, programas e objetos tipo **Smalltalk**. Os trabalhos desse grupo podem ser divididos em duas classes.

Os pesquisadores da primeira classe colocam mais ênfase no desenvolvimento e acumulação de bibliotecas de componentes. A pesquisa de [LANE 1984] objetiva criar uma biblioteca de componentes de código (subrotinas) úteis sem grandes preocupações com especificações ou formalismos de composição.

Os pesquisadores da segunda classe buscam desenvolver princípios de organização e composição pelos quais os componentes são organizados formando programas. Assim, um novo sistema é construído a partir da composição de programas existentes, que por sua vez também foram construídos a partir da composição de componentes existentes. Um exemplo dessa classe é um mecanismo baseado no sistema UNIX [KERN 19841, que permite a construção de programas complexos a partir de programas mais simples, através da conexão da saída de um programa às entradas de um outro programa.

II.2.2 – TECNOLOGIAS DE GERAÇÃO

Este grande grupo também recebe o nome de tecnologias de reutilização por padrões ou programação automática. Os sistemas baseados nesta tecnologia não são tão fáceis de serem caracterizados. Não se tem mais componentes passivos, que são reunidos para formar um programa. Neste grupo, os componentes são elementos ativos que geram o programa. As estruturas resultantes frequentemente guardam uma pequena relação com os padrões de programas originais ou com outras estruturas geradas a partir do mesmo padrão. Os efeitos dos componentes reutilizados no programa alvo tendem a ser mais globais e difusos do que os efeitos da reutilização por composição e com isso não se consegue apontar para uma parte bem definida do programa alvo e identificá-la como um componente reutilizado. A seguir será descrita uma classificação dos diferentes tipos de tecnologias de geração.

II.2.2.1 – SISTEMAS BASEADOS EM LINGUAGEM

Os sistemas geradores baseados em linguagem enfatizam o uso de uma linguagem de largo espectro para descrever o desenvolvimento do software, desde a especificação até a sua implementação final. A ênfase principal está na notação que é usada para descrever o sistema alvo. Os requisitos, representados na linguagem por construções de alto nível, são transformados por refinamentos em primitivas no nível de implementação. Em um dado instante, o sistema que está sendo refinado poderá incluir declarações de diversos níveis de abstração [FARI 1991].

Os sistemas baseados em linguagem variam em um largo espectro desde linguagens de muito alto nível, que são linguagens de finalidade geral, até linguagens orientadas ao problema, que são linguagens de finalidade específica. A característica principal que diferencia as duas abordagens é o grau no qual as construções da linguagem contêm informações específicas sobre o domínio do problema [BIGG 1984]. Com isso, as linguagens orientadas ao problema são mais restritas, mas oferecem um potencial de reutilização bem maior.

A linguagem MODEL [CHNG 1984] exemplifica essa categoria de sistemas.

II.2.2.2 – SISTEMAS BASEADOS EM TRANSFORMAÇÃO

Os sistemas geradores baseados em transformação recebem como entrada uma especificação de alto nível e vão aplicando uma seqüência de transformações nesta especificação até se obter uma implementação de baixo nível. Cada etapa dessas gera refinamentos em um nível cada vez mais concreto, mas que se mantêm equivalentes logicamente.

Nestes sistemas, os componentes reutilizáveis são padrões que existem dentro das regras de transformação [BIGG 19841. O sistema de transformação funciona da seguinte maneira : ele examina as especificações do sistema alvo para tentar descobrir padrões contidos nas regras; se encontrar, a regra associada realiza uma transformação. Este processo se repete até que uma condição de término é encontrada.

Em [CHEA 19841, encontramos um exemplo de um sistema de transformação como uma metodologia de desenvolvimento de programas.

II.2.2.3 – SISTEMAS BASEADOS EM DEDUÇÃO

Segundo [RICH 19881, o problema de sintetizar um programa que satisfaça uma determinada especificação é formalmente equivalente a descobrir uma prova construtiva da verificação lógica de uma especificação. Esta idéia fundamental forma a base da abordagem dedutiva da programação automática. Em princípio, qualquer método de dedução – resolução, dedução natural, etc – pode ser usado para dar suporte à programação automática, mas na prática nenhum desses métodos foi capaz até hoje de provar os tipos de teoremas complexos necessários para sintetizar programas reais.

II.2.2.4 – GERADORES DE APLICAÇÃO

Os geradores de aplicação geralmente embutem uma grande quantidade de informação específica sobre o domínio da aplicação dentro do próprio gerador de aplicação, ou seja, os componentes reutilizáveis são padrões de código que encontram-se embutidos dentro do próprio gerador [BIGG 1984].

Um gerador de aplicação é um pacote de **software** que é projetado para ajudar usuários finais a construir aplicações em um determinado domínio. Comparado com as categorias anteriores, estes pacotes possuem um conhecimento muito específico sobre a área de aplicação para o qual os programas vão ser escritos. Isto permite que a linguagem de especificação seja muito precisa. É possível, também, considerar um gerador de aplicações como sendo o resultado de uma análise de domínio. Quanto mais restrito for o domínio da aplicação, o benefício com o uso de geradores de aplicação será maior e mais rápido [HORO 1984]. Dependendo da quantidade de lógica que deve ser desenvolvida fora do gerador de aplicação, o potencial de reutilização varia entre 60% e 90% [BIGG 1987].

O sistema DRACO, desenvolvido por Neighbors [NEIG 1984] é um exemplo de gerador de aplicação, embora também se possa argumentar que ele também se enquadra nas categorias baseadas em linguagem e em transformação. Um outro exemplo é o LANCE [KOGU 1991], desenvolvido no Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro, que é um gerador de aplicações que produz um sistema a partir do conjunto de dados que o descreve.

Uma classe particular de geradores de aplicação é constituído das denominadas linguagens de quarta geração. Nestas, o usuário geralmente fornece uma especificação de forma interativa e é gerado um programa executável, que tem acesso a uma base de dados e imprime alguns relatórios. Estas linguagens,

quando aplicáveis, permitem um grande aumento na produtividade de desenvolvimento de software, mas possuem a desvantagem de não gerar programas em linguagens tradicionais, o que inviabiliza o uso de soluções conjugadas, como a utilização de escapes.

O escape é um componente do gerador que permite que alguma coisa seja expressa na linguagem de implementação em questão. O escape acrescenta poder e flexibilidade ao gerador, aumentando muito o seu alcance. O escape também degrada a confiabilidade do gerador e a facilidade de leitura da linguagem de especificação. Por isso, sua utilização deve ser minimizada.

11.3 - A CONSTRUÇÃO DE GERADORES DE APLICAÇÃO

Os geradores de aplicação são ferramentas de apoio ao desenvolvimento de software que, além de reduzir os custos e aumentar a produtividade, ajudam a melhorar a qualidade dos sistemas produzidos. Essencialmente, um gerador de aplicação é um utilitário que, a partir de uma especificação em alto nível de um problema implementável, transforma automaticamente essa especificação na implementação do problema. Os geradores de aplicação são mais conhecidos e têm obtido sucesso no domínio de Sistemas de Informação [MASI 1991].

Em [CLEA 19881, encontramos uma análise de algumas vantagens e desvantagens dos geradores de aplicação. Com o uso de um gerador de aplicação a fase de manutenção fica facilitada, devido à modularidade e à uniformidade do código gerado . Os erros de programação ficam reduzidos, ao invés de se modificar diretamente o código, basta modificar a especificação de entrada e fornecê-la novamente ao gerador.

A especificação, por não conter detalhes de implementação, é mais fácil de ser escrita, lida e modificada. Por isso, é possível que as aplicações sejam geradas e mantidas por pessoas que não saibam programar. Uma outra vantagem dos geradores de aplicação é que eles, por acelerarem o processo de codificação, facilitam a construção de protótipos e o teste de especificações alternativas. A implementação de padrões fica facilitada devido à criação de uma **interface** padrão ou de um formato de saída.

Por outro lado, os geradores de aplicação também apresentam algumas dificuldades. Eles podem ser usados efetivamente em poucas situações e são difíceis de ser construídos, pois requerem linguagens de especificação e interfaces com o usuário cuidadosamente projetadas, um conhecimento muito grande do domínio da aplicação e a habilidade de projetar unidades de software genéricas e confiáveis. O reconhecimento de que um gerador de aplicação pode ser usado é difícil e muitas vezes

ocorre tarde no ciclo de vida, quanto então já não existe motivação para refazer o desenvolvimento.

Um outro problema é a dificuldade da organização enfrentar as questões que os geradores de aplicação impõem. Como e quando encaixar os geradores de aplicação no processo de desenvolvimento de **software** habitual? Quem deve realizar isso? Se os geradores não estão disponíveis, deve-se ou não construí-los? Como eles afetam o cronograma? É difícil comparar os benefícios a longo prazo com os custos a curto prazo de se construir geradores de aplicação.

Embora existam vários exemplos de geradores de aplicação em diferentes áreas, há pouca semelhança entre eles na forma como é realizada a tradução da especificação de entrada para o produto final. Isso faz com que o desenvolvimento de um gerador para uma nova área seja um novo esforço, sem a existência de meios para facilitar e agilizar esse desenvolvimento. É necessário, então, que se tenha uma forma mais organizada para a construção dos geradores de aplicação e é disso que trataremos nos próximos itens.

II.3.1 – A ESTRUTURA DE UM GERADOR DE APLICAÇÃO

Em seu artigo sobre geradores de programa, Luker e Burns [BURN 1986] descrevem os componentes principais necessários para quase todos os geradores, como se vê na figura 2.

O módulo de diálogo interage com o usuário para determinar os requisitos da aplicação e produz uma especificação da aplicação. O objetivo principal desse módulo é auxiliar o usuário a formular uma especificação completa e consistente do problema que o gerador vai resolver.

O módulo intermediário transforma a especificação em um meta-programa, escrito em uma meta-linguagem (que pode ser uma linguagem de alto nível existente). O meta-programa consiste de uma definição concisa das operações que devem ser realizadas pela aplicação gerada. O meta-programa deve ser facilmente lido tanto pelo usuário quanto pela máquina.

Finalmente, o módulo gerador de código produz a aplicação a partir do meta-programa.

Nem todos os componentes descritos são necessários para todos os geradores e alguns geradores têm ainda módulos adicionais para finalidades específicas.

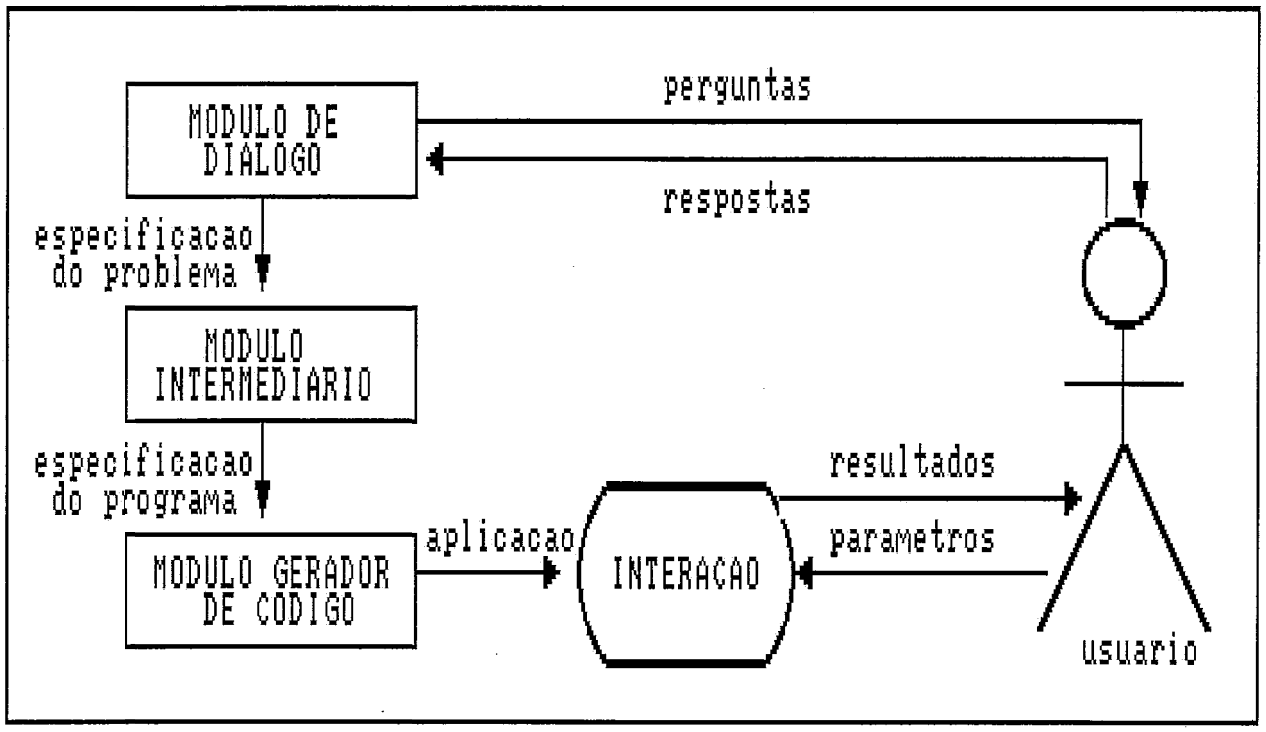


FIGURA 2 - ESTRUTURA DE UM GERADOR DE APLICACAO

II.3.2 - UMA ABORDAGEM PARA A CONSTRUÇÃO DE GERADORES

Em um ambiente com o suporte de geradores de aplicação surgem dois novos personagens: o analista de domínio e o projetista de domínio [CLEA 1988]. A figura 3 mostra como ficam as várias funções num ambiente típico com o suporte de geradores de aplicação.

O analista de domínio é quem especifica os requisitos do gerador, analisando as necessidades e os requisitos de uma série de problemas na área de interesse. O projetista de domínios toma essa especificação e a implementa no gerador. O projetista de sistemas, agora com o auxílio do gerador de aplicação, recebe a especificação do sistema e a transforma numa implementação.

Seria ideal que o trabalho do projetista de sistema se reduzisse apenas a fornecer corretamente a especificação para o gerador de aplicação, produzindo o sistema por completo. Entretanto, na prática, os geradores de aplicação geralmente criam apenas partes de um sistema, deixando o resto do trabalho para o projetista. Essas partes são chamadas de produtos da aplicação, que podem ser estruturas de dados, subrotinas ou segmentos de código.

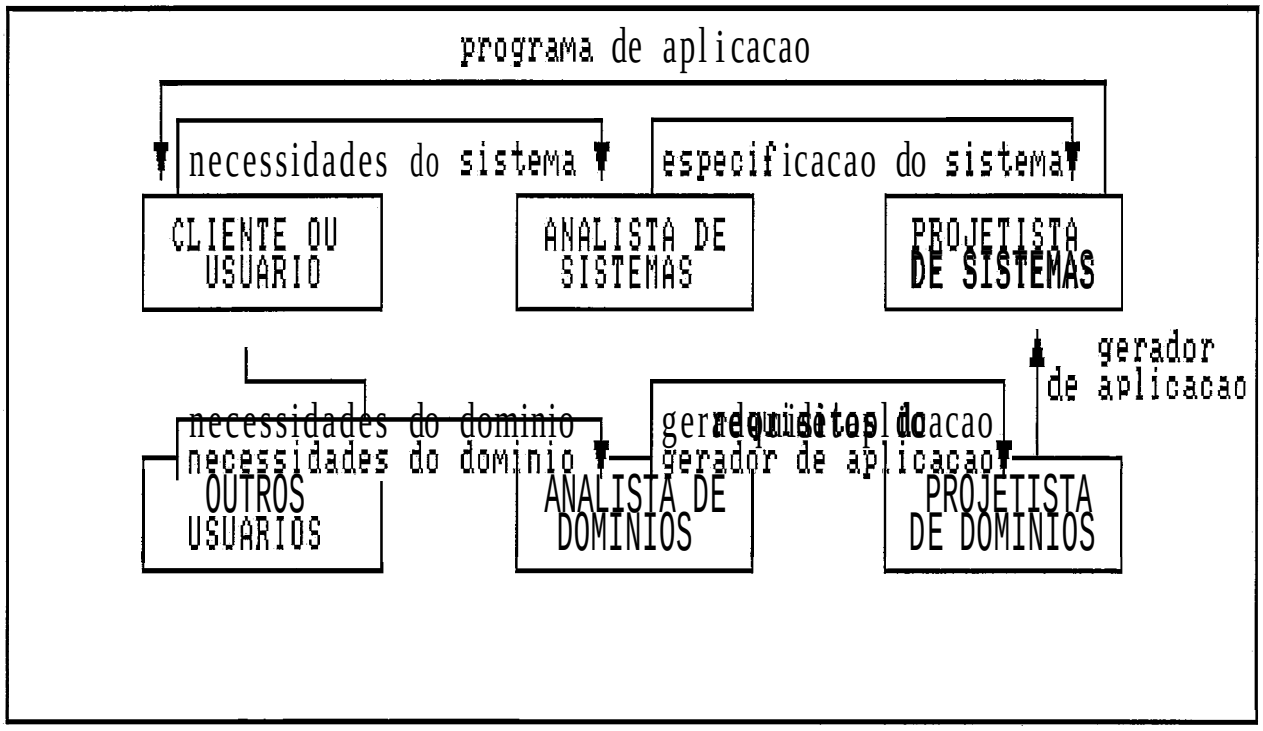


FIGURA 3 - RELACIONAMENTO ENTRE AS FUNCOES

[CLEA 19881 sugere ainda uma abordagem metódica para a construção de geradores de aplicação:

1. **Reconhecimento de domínios.** A dificuldade dessa etapa consiste no reconhecimento de padrões. Padrões podem ocorrer no nível de código ou em níveis mais altos; como programas, projetos e arquiteturas similares.

2, Definição de limites do domínio. Esta etapa vai determinar o alcance do gerador, isto é, quais aspectos vão ser incluídos ou excluídos. A definição de limites vai determinar quais problemas o gerador vai ser capaz de resolver. Como em qualquer projeto, é preciso fazer um balanceamento entre alcance, custo e eficiência.

3. Definição de um modelo base. Um gerador tem mais chances de ser abrangente, consistente e completo se ele é baseado num modelo matemático. O modelo provê a base para definir a semântica do domínio consistentemente. Alguns exemplos de modelos são conjuntos, grafos dirigidos e sistemas de lógica formal.

4. Definição das partes variáveis e invariáveis. A parte variável é algo sob o controle do projetista do sistema. A

parte invariável é algo que não pode ser modificado. As partes variáveis geralmente correspondem à especificação do sistema. As partes invariáveis (o como fazer) são geralmente idéias que se assumem sobre o domínio ou a implementação; são detalhes de projeto que os usuários preferem não se preocupar. Uma outra forma de informação variável é o **escape,** já explicado anteriormente.

5. Definição da especificação de entrada. A especificação de entrada pode ter vários formatos e a escolha do melhor vai depender da aplicação. Alguns dos formatos mais comuns são:

– **Uma linguagem ou notação textual.** Este método é útil quando existe muita informação ou quando as especificações incluem expressões complexas ou usam **escapes.**

– **Diálogos e abordagens interativas.** Este método deve ser empregado quando a maior parte da informação da aplicação pode ser selecionada de um cardápio.

– **Diagramas.** Esta abordagem é recomendada quando a especificação possui a forma pictorial.

6. Definição de produtos. O produto de um gerador pode ser qualquer coisa: um programa, documentação ou até mesmo dados de teste.

Estas seis primeiras etapas são o trabalho do analista de domínio.

7. Implementação do gerador. A última etapa é a mais fácil delas. O projetista de domínio determina como implementar o gerador a partir dos requisitos especificados pelo analista de domínio.

É importante ressaltar que os geradores tendem a criar programas que normalmente não seriam feitos manualmente. Estes programas gerados acabam executando novos caminhos do compilador e por isso às vezes descobrem novos erros ou limitações do compilador.

Como os programas gerados não devem ser modificados e são raramente lidos, os geradores não costumam levar em conta o estilo de programação. Assim, a maior parte dos programas gerados possuem um formato ruim, incluindo subrotinas monolíticas, constantes misteriosas, comandos não estruturados como **gotos,** variáveis pobremente denominadas e uma formatação fraca.

Considerando esses problemas comuns nos programas gerados e o fato das ferramentas de depuração padrão trabalharem a nível de código e não a nível de especificação, conclui-se que vale a

pena investir nos geradores para que estes gerem programas mais fáceis de ler.

A figura 4 mostra o processo básico necessário para desenvolver um sistema com a utilização de um gerador de aplicação.

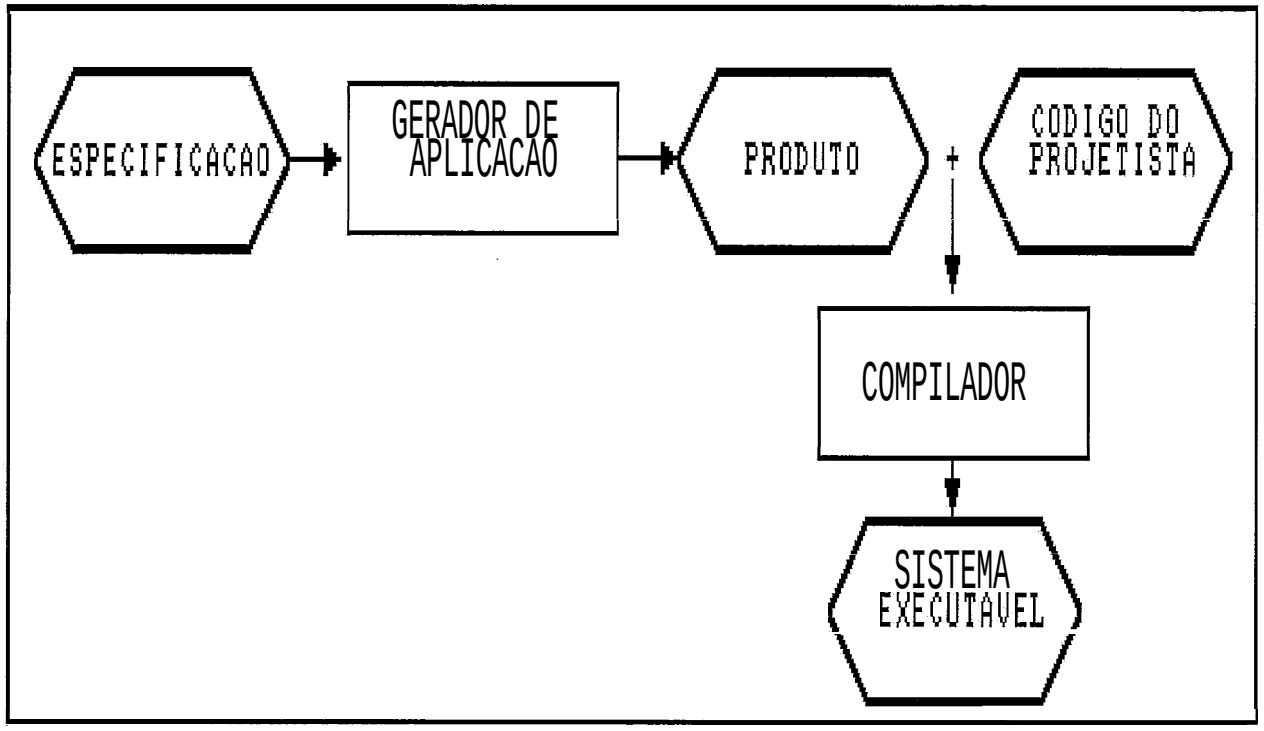


FIGURA 4 - PROCESSO DE DESENVOLVIMENTO USANDO UM GERADOR

II.3.3 - COMPONENTES BÁSICOS DOS GERADORES DE APLICAÇÃO

Encontramos em [HORO 19851 uma descrição dos componentes básicos dos geradores de aplicação que estão presentes na maioria dos sistemas atualmente disponíveis.

. **SGBD.** Os geradores de aplicação têm o seu próprio SGBD. Geralmente, o modelo de dados escolhido é hierárquico ou relacional. O banco de dados possui um dicionário de dados, que contém informações fornecidas interativamente pelo usuário sobre um determinado banco de dados, como por exemplo tipos de campos, nomes de campos, nomes dos arquivos, chaves de acesso dos arquivos e relacionamentos entre os arquivos.

Além do dicionário de dados, o banco de dados possui também os arquivos de dados, cujos dados serão manipulados pelo usuário através de uma linguagem de manipulação de dados.

. **Gerador de relatório.** Este componente provê facilidades para a extração de dados dos arquivos e a sua formatação para relatórios. Os geradores de relatório são considerados linguagens não procedurais, pois o usuário especifica o que deve ser realizado, sem precisar descrever como deve ser realizado.

. **Pacote gráfico.** Na realidade, este componente é um tipo especial de gerador de relatório que produz um relatório gráfico. O usuário precisa informar qual o tipo de gráfico que deve ser utilizado, como por exemplo histogramas. As técnicas gráficas são um meio poderoso para que se possa entender dados complexos.

. **Procedimentos de manipulação.** Este componente deve ser capaz de realizar as modificações necessárias no banco de dados. É necessário que as seguintes operações possam ser executadas: incluir registros, excluir registros, alterar dados dos registros, acessar e listar registros, verificar restrições de segurança e verificar restrições de integridade.

II.4 – PERSPECTIVAS DA PROGRAMAÇÃO AUTOMÁTICA

Alguns pesquisadores e gerentes de **software** têm aguardado ansiosamente o dia em que os programas não serão mais escritos por programadores e sim por outros programas. Embora um número de desenvolvimentos importantes esteja aparecendo em esforços de pesquisa e em sistemas disponíveis comercialmente, as expectativas em relação ao potencial da programação automática são frequentemente baseadas em uma visão idealizada da realidade e são provavelmente inatingiveis.

Existem vários mitos em relação à programação automática [RICH 1988]:

. **Mito :** sistemas de programação automática não precisam conhecer o domínio da aplicação. **Realidade :** assim como um programador humano tem que conhecer a linguagem em que vai programar e precisa ter conhecimentos do domínio da aplicação, o sistema de programação automática precisa ser um especialista em geração de programas e também no domínio da aplicação.

. **Mito :** sistemas de programação automática podem ser simultaneamente: i) orientados para o usuário, ii) de

finalidade geral e iii) totalmente automatizados. Como conseqüência do fato do sistema precisar conhecer o domínio da aplicação, um sistema de finalidade geral precisaria ser especialista em todos os domínios de aplicação e isto é atualmente impossível com os atuais conhecimentos de inteligência artificial. **Realidade :** todas as abordagens atuais de programação automática enfatizam dois dos aspectos acima, sacrificando o terceiro.

. **Mito :** especificação de requisitos pode ser completa. **Realidade :** especificação de requisitos de qualquer complexidade é uma aproximação incompleta e o sistema de programação automática deve ser orientado para assumir conclusões razoáveis sobre propriedades não especificadas ao invés de somente tentar satisfazer propriedades especificadas.

. **Mito :** programação é um processo serial. Primeiro, o usuário cria um requisito. Depois, o sistema de programação automática faz um programa. **Realidade** : a programação manual e a programação automática são processos interativos e os requisitos evoluem a partir de protótipos e versões iniciais do sistema.

. **Mito** : não haverá mais programação. **Realidade** : os usuários finais se tornarão programadores, aliando seu conhecimento do domínio da aplicação com conhecimentos de computação específicos para sua aplicação.

Em outras palavras, existem complexidades no processo de escrever um programa que não podem ser deixadas de lado. A programação automática não é a panacéia universal para todos os males da programação. Segundo [RICH 1988], é preciso ter uma perspectiva realista de que "os sistemas de programação automática do futuro serão mais parecidos com aspiradores de pó do que com fornos auto-limpantes. Com o forno auto-limpante, ao se decidir que se quer o forno limpo, é necessário apertar um botão. Com o aspirador de pó, a produtividade é bastante aumentada, mas ainda se tem muito trabalho a fazer".

A pesquisa acadêmica sobre programação automática tem enfatizado o desenvolvimento de técnicas que possam suportar sistemas de finalidade ampla e totalmente automatizados. Infelizmente, os resultados dessa pesquisa parecem indicar somente progresso a longo prazo. Na área comercial, os objetivos têm sido mais modestos e tem havido alguns avanços significativos em algumas categorias : sistemas de consultas a banco de dados, linguagens de quarta geração, geradores de programa e linguagens de muito alto nível para prototipação.

CAPÍTULO III - MODELOS DE DADOS

III.1 - INTRODUÇÃO

Um modelo de dados é uma coleção de conceitos que pode ser usada para descrever um conjunto de dados e as operações para manipular esses dados. Os conceitos em um modelo de dados são tipicamente construidos usando-se mecanismos de abstração e são descritos através da lingüística e de representações gráficas. Modelos de dados conceituais são ferramentas para representar a realidade em um alto nível de abstração. Usando modelos conceituais, pode-se construir uma descrição da realidade que é fácil de entender e interpretar : o esquema conceitual. Modelos de dados lógicos suportam descrições dos dados que podem ser processadas por um computador, como os modelos hierárquico, CODASYL (rede) e relacional. Os modelos lógicos são facilmente mapeados para a estrutura física do arquivo [CERI 19921.

A arquitetura de um sistema de banco de dados, conforme proposta pela ANSI / SPARC [ANSI 19751, pode ser dividida em três níveis :

- Externo ou visão : é aquele que descreve o mundo real segundo a visão de cada usuário individualmente. Pode haver diversas visões para um mesmo banco, não necessariamente disjuntas.

- Conceitual : consiste na representação de todas as informações armazenadas no banco. O objetivo é representar quais são os dados que compõem o banco e não os detalhes de como estes dados estão armazenados.

- Interno ou físico : este nível retrata a forma física final com que os dados são armazenados em meios computacionais.

Nesta arquitetura, modelos conceituais são linguagens para implementar o nível conceitual e a preocupação maior tem ficado por conta da implementação a partir da modelagem do esquema interno. A modelagem do esquema conceitual, entretanto, é de vital importância e deve refletir o mundo real de forma simples, completa e correta.

Muitas propostas de modelos de dados têm sido feitas visando suportar aspectos semânticos de representação de informação. Fazem parte destas discussões os Modelos Semânticos surgidos ao longo da década de 70. Como exemplo temos o modelo entidade-relacionamento [CHEN 1976] e o modelo relaciona1 estendido [CODD 1979].

III.2 – QUALIDADES DE MODELOS E ESQUEMAS CONCEITUAIS

Modelos conceituais devem ser boas ferramentas para representar a realidade; por isso, segundo [CERI 1992], devem possuir as seguintes qualidades :

– **Expressividade** : os modelos conceituais diferem entre si na escolha e no número de diferentes estruturas de modelagem que possuem. Em geral, a disponibilidade de uma grande variedade de conceitos possibilita uma representação mais abrangente do mundo real; por isso, modelos que são ricos em conceitos são também muito expressivos.

– **Simplicidade :** um modelo conceitual deve ser simples **,** de maneira que um esquema construído usando tal modelo seja facilmente compreensível para os projetistas e usuários da aplicação. É importante ressaltar que simplicidade e expressividade são objetivos às vezes conflitantes.

– **Minimidade** : esta propriedade é atingida se todo conceito no modelo tem um significado distinto de qualquer outro conceito.

– **Formalidade :** Todos os conceitos do modelo devem ter uma interpretação única, precisa e bem definida. Conceitos formais podem ser matematicamente manipulados.

– **Representação gráfica :** deve ser fácil de ler e completo graficamente. Um modelo é fácil de ler se cada conceito é representado por um símbolo gráfico que é claramente distinto de todos os outros símbolos gráficos. Um modelo é completo graficamente se todos os seus conceitos possuem uma representação gráfica.

Um esquema é considerado adequado conceitualmente ou de alta qualidade se satisfaz os seguintes critérios [RISH 1988] :

– O esquema descreve os conceitos do mundo real naturalmente, isto é, o usuário pode traduzir idéias facilmente em ambas direções entre os conceitos do esquema e os conceitos naturais da aplicação no mundo real.

– O esquema não contém redundância. Se não for possível eliminar a redundância completamente, deve-se ligá-la a restrições de integridade, para que quando estas restrições forem implementadas, o usuário seja forçado a atualizar todos os fatos relacionados simultaneamente.

- O esquema não impõe restrições de implementação.

- O esquema é capaz de representar o máximo possível de restrições de integridade.

- O esquema é flexível : se mudanças nos conceitos do mundo real acontecerem, o esquema não deve precisar sofrer mudanças drásticas.

- O esquema é conceitualmente mínimo. Ele não deve representar conceitos que são irrelevantes na aplicação do mundo real.

III.3 - O MODELO ENTIDADE-RELACIONAMENTO

III.3.1 - INTRODUÇÃO

O Modelo Entidade-Relacionamento constitui a base do ambiente de projeto de bancos de dados fornecido pelo UniversiData. Sua compreensão é fundamental tanto para a definição do banco, quanto para o desenvolvimento de aplicações que utilizam o mesmo.

O Modelo Entidade-Relacionamento foi introduzido por Peter Chen [CHEN 1976] e foi um dos primeiros trabalhos produzidos na área de modelagem semântica de dados. O Modelo E-R foi criado para facilitar o projeto de banco de dados por permitir a especificação do esquema de uma organização [CHEN 1977]. O esquema de uma organização representa a visão do conjunto de dados de uma organização inteira e é independente de considerações de armazenamento ou eficiência. O trabalho original de Chen ignorava aspectos de integridade e manipulação do banco, sendo por isso questionado quanto a sua validade como um modelo de dados, visto que um modelo de dados não deve ser restrito apenas a definição de suas estruturas de dados [CODD 1981]. Diversas extensões foram propostas ao modelo original, de forma a abranger aspectos de integridade e manipulação do banco, além da definição de novos componentes estruturais. Várias interpretações diferentes também foram fornecidas por diversos autores. [DATE 19861 considera o modelo E-R como uma camada acima do modelo relaciona1 básico, ambos apresentando a mesma estrutura de dados e os mesmos operadores, diferindo apenas quanto as restrições de integridade. Realmente, é possível um mapeamento dos componentes do modelo E-R para uma representação em forma de tabelas [KORT 19861 [ULLM 19801. Um enfoque mais formal do Modelo-ER pode ser encontrado em [SILV 1989].

III.3.2 – COMPONENTES BÁSICOS

A seguir será dada uma breve descrição dos componentes do modelo:

– **Entidade :** é uma representação abstrata de um objeto do mundo real que pode ser distintamente identificável. Entidades são representadas graficamente no MER por um retângulo.

– **Relacionamento :** são associações entre elementos dos conjuntos de entidades. O grau de um relacionamento indica o número de entidades participantes. A representação gráfica se faz através de um losango.

– **Atributo** : são usados para descrever as características de um objeto. Um atributo pode ser multivalorado ou possuir somente uma ocorrência. Uma entidade deve possuir ao menos um atributo que a caracterize.

– **Auto-relacionamento ou anel** : são relacionamentos binários conectando uma entidade a essa mesma entidade.

– **Papel :** é a função que uma entidade desempenha em um relacionamento. Em um auto-relacionamento, por exemplo, é necessário que sejam definidos os papéis.

Além dos componentes acima, o MER possui outros elementos que são caracterizados como propriedades do esquema e são, portanto, considerados restrições de integridade:

– **Entidade Fraca :** é um tipo especial de entidade cuja identificação e existência dependem de uma outra entidade, via um relacionamento. Sua representação gráfica é feita através de dois retângulos, como se vê na figura 5.

– **Chaves :** é um conjunto de atributos que identifica unicamente uma instância de uma entidade ou relacionamento.

– **Cardinalidade :** é um tipo de restrição de integridade importado pelo modelo. Permite a especificação do número mínimo e máximo de instâncias de uma entidade que podem participar de um determinado relacionamento, através de uma função papel. O número mínimo e máximo de instâncias pode ser indicado através de um valor explícito ou de um literal (n ou m) indicando um número qualquer maior que zero. Assim, na figura 6, observa-se que um professor pode ter no mínimo zero e no máximo **n** orientandos, ao passo que cada aluno deve ter no mínimo um e no máximo também um professor orientador.

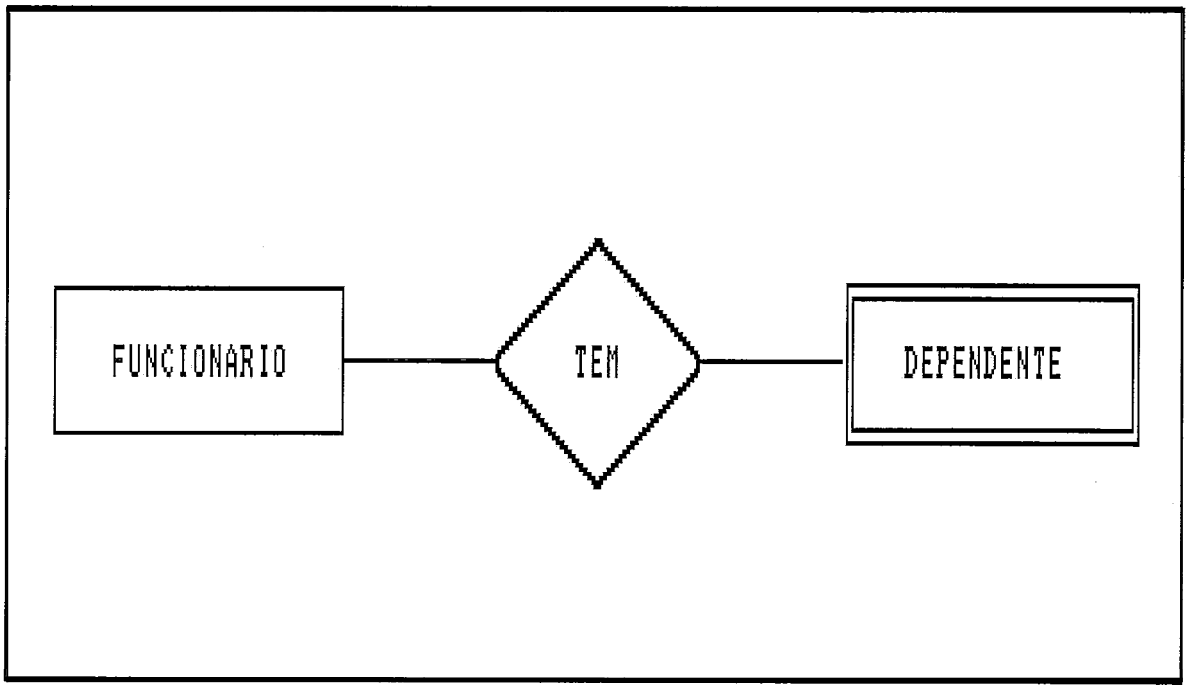


Figura 5 - ENTIDADE FRACA

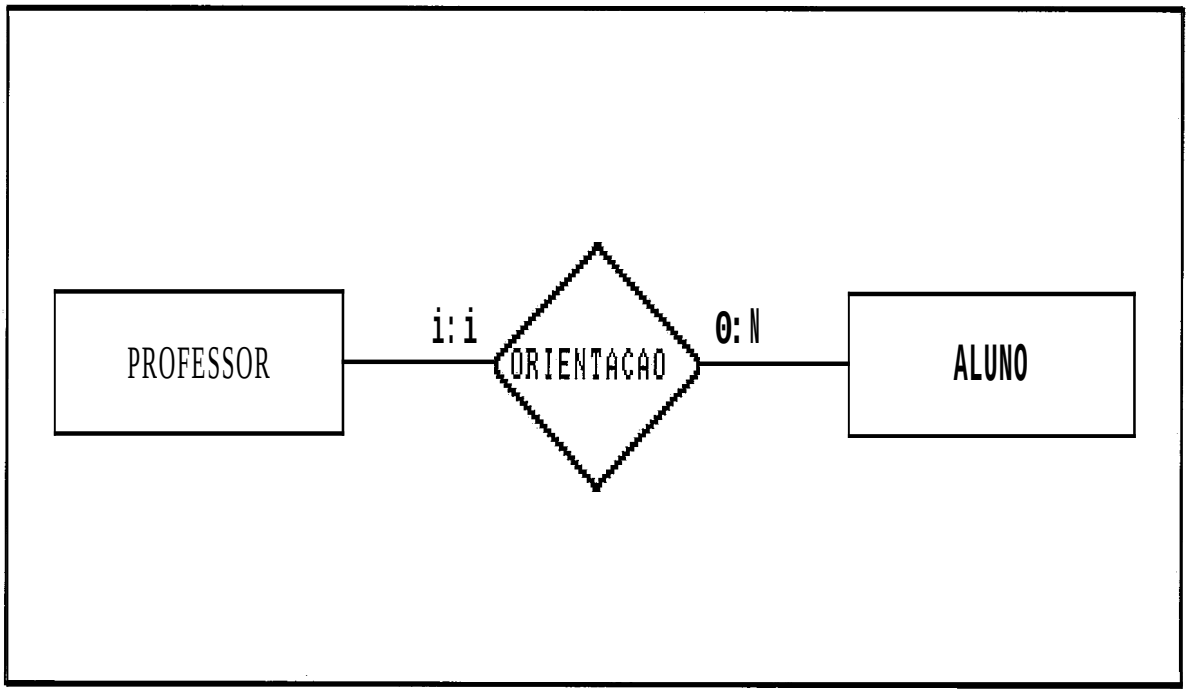


Figura 6 - CARDINALIDADE

III.3.3 – EXTENSÕES

Muitas propostas de extensões têm sido feitas ao Modelo E-R proposto por Chen. Serão apresentadas aqui somente as extensões que foram utilizadas no projeto UniversiData [Silv 1990] :

– **Generalização/Especialização** : a generalização é um mecanismo de abstração através do qual diferenças entre objetos similares são ignoradas para propiciar a concepção de uma classe de mais alta ordem, em que as similaridades podem ser enfatizadas [TAKA 19891. Especialização é o processo inverso, onde se pode ir do geral para o específico, gerando sub-entidades ou subtipos. A entidade que é subtipo herda os atributos de seu supertipo [PECK 19881. A vantagem deste mecanismo é a eliminação da redundância na definição de entidades semelhantes. Podem existir vários níveis de generalização, mas uma entidade só pode ser sub-entidade de uma única super-entidade. Sua representação é vista na figura 7.

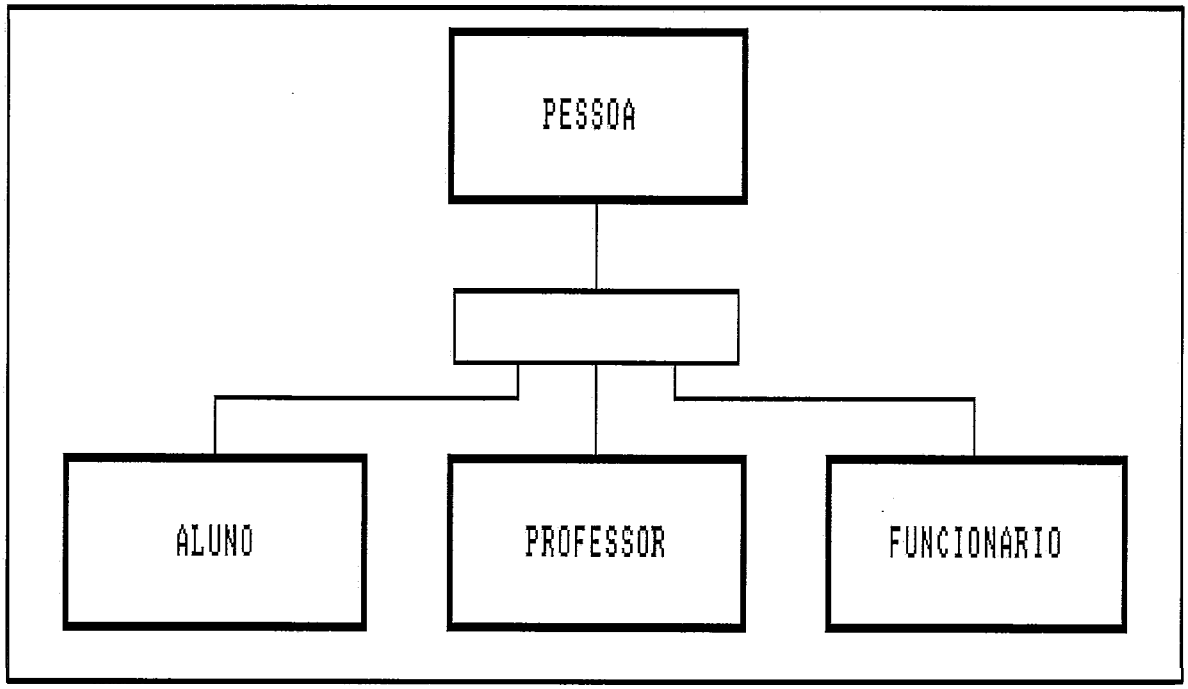


FIGURA 7 - GENERALIZACAO/ESPECIALIZACAO

– Ordens : o conceito de ordem é similar ao de chave, porém a ordem admite duplicação. O acesso através de uma ordem não é suportado pelo Modelo E-R em sua definição original, pois este não provê suporte para implementações físicas. O acesso ordenado, entretanto, é largamente utilizado durante a programação e pode ser obtido a partir de :

a) um domínio, que é um conjunto de elementos de entidades/relacionamentos que devem ser ordenados;

b) uma expressão de seleção, que designa quais elementos do domínio devem aparecer na ordem final;

c) os determinantes da ordenação, que são atributos ou outros termos do domínio da ordenação que determinam o critério de ordenação a ser seguido.

– Entidade Associativa : é um componente utilizado para representar o conceito de agregação [SMIT 1977]. A agregação é um mecanismo de abstração através do qual relacionamentos são vistos como entidades. Assim, consegue-se expressar relacionamentos entre relacionamentos. Segundo [SETZ 1989], não há muito sentido em se falar de atributos de uma agregação, pois estes coincidem com os atributos do relacionamento que ela engloba. A utilização de agregação evita a formação de relacionamentos de grau maior do que dois, que muitas vezes são ambíguos. A entidade associativa ora possui o comportamento de uma entidade (participando de relacionamentos), ora o comportamento de um relacionamento (associando entidades). É possível definir tanto chaves quanto ordens para entidades associativas. Observa-se um exemplo de entidade associativa na figura 8.

III.4 – RESTRIÇÕES DE INTEGRIDADE

III.4.1 – RESTRIÇÕES

Segundo [TSIC 1982], existem algumas propriedades adicionais dos dados sendo modelados que não conseguem ser representadas por atributos, entidades ou relacionamentos. Estas propriedades podem ser expressas como restrições adicionais aos valores dos dados e/ou a como os dados podem ser relacionados (estruturados). Na modelagem de dados, restrições são úteis quando são genéricas, isto é, quando podem ser definidas e aplicadas a um conjunto de objetos e não a uma instância particular de um objeto. As restrições são necessárias em um modelo de dados por razões semânticas e de integridade. Em termos de semântica, as restrições permitem que os esquemas reflitam mais precisamente a situação do mundo real. Em termos

de integridade, as restrições permitem que o SGBD restrinja os possíveis estados do banco de dados que podem ser gerados a partir de um determinado esquema.

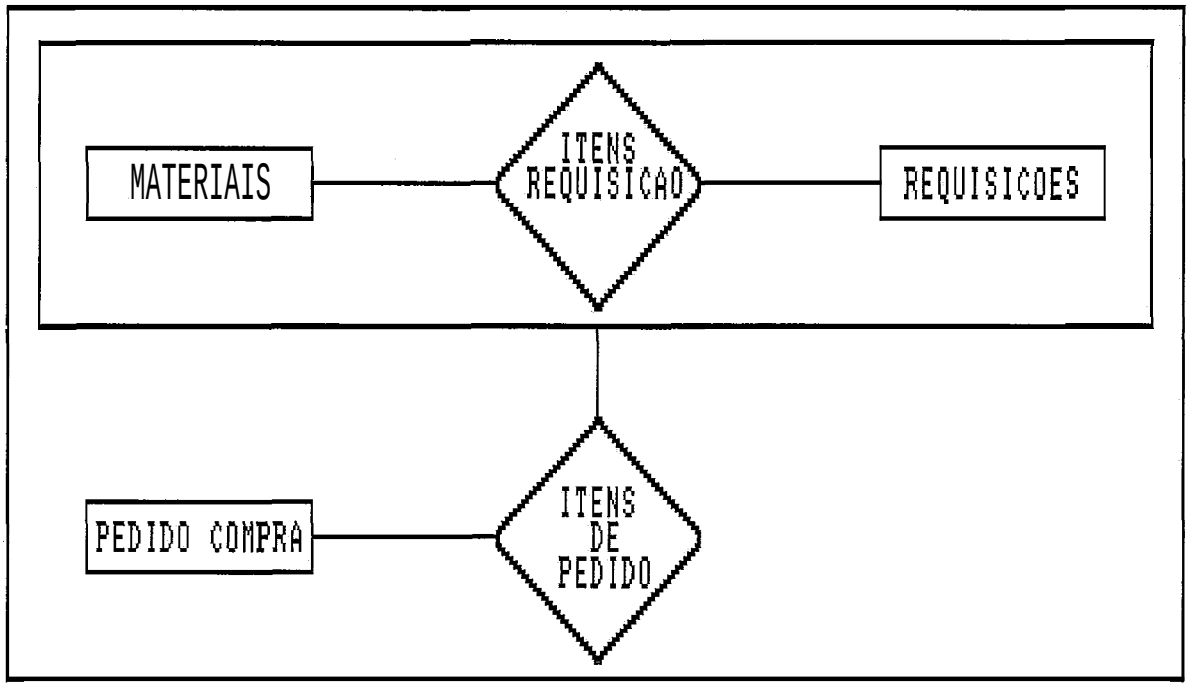


FIGURA 8 - ENTIDADE ASSOCIATIVA

III.4.2 – RESTRIÇÕES DE INTEGRIDADE NO MODELO RELACIONAL

Todas as inclusões, alterações e exclusões realizadas nas relações são restritas pelas seguintes regras:

– **Integridade de Entidade :** nenhum atributo participante da chave primária de uma relação pode ter valor nulo.

– **Integridade Referencial :** se uma relação R2 possui uma chave estrangeira CE correspondendo à chave primária CP de alguma relação R1, então todo valor de CE em R2 deve ser totalmente nulo ou ser igual ao valor de CP em alguma tupla de R1.

Qualquer estado do banco de dados que não satisfaça às duas regras acima é por definição incorreto. A primeira solução para evitar um estado incorreto do banco de dados é rejeitar qualquer operação que resultaria em um estado ilegal. Uma outra solução é o SGBD aceitar a operação e realizar algumas operações

adicionais de modo a garantir que o resultado final seja um estado correto.

Na detalhada descrição do Modelo Relaciona1 Estendido (RM/T) encontrada em [CODD 1979] e [DATE 1985] são introduzidas outras seis novas regras de integridade que detalham e quase substituem totalmente a regra de integridade referencial.

III.4.3 – RESTRIÇÕES DE CARDINALIDADE NO MODELO E-R

[LENZ 1983] propõe um classe particular de restrições de integridade, denominadas restrições de cardinalidade e propõe estruturas adequadas para expressá-las em um Modelo Entidade-Relacionamento enriquecido. Há três tipos de restrições de cardinalidade :

- Cardinalidade mínima (Cmin) e cardinalidade máxima (Cmax) de uma entidade representam o número mínimo e máximo de instâncias da entidade ou relacionamento que podem existir no banco de dados.

- Cmin e Cmax de um conjunto de entidades em relação a um outro conjunto de entidades no contexto de um relacionamento representam o número mínimo e máximo de instâncias de um conjunto de entidades que podem se relacionar a uma instância de um outro conjunto de entidades através de um determinado relacionamento.

- Cmin e Cmax de um atributo representam o número mínimo e máximo de valores do domínio correspondente que podem ser associados a uma instância da entidade correspondente.

O segundo tipo de restrições de cardinalidade (RC) é chamado de RC relativas. Este é um tipo muito utilizado e apresenta as seguintes definições:

- Seja Z o conjunto de entidades E1,...,Em envolvidas em R;

- Sejam X,Y dois subconjuntos disjuntos de Z e W = Z-(X U Y).

- Definição 1. Cmax de X, com respeito a Y, no contexto de R, é uma expressão da seguinte forma: Cmax[R(X/Y)] = K, onde K > 0 e impõe que o número de instâncias diferentes de X relacionadas com qualquer instância y de Y por meio de R é menor que ou igual a K.

- Definição 2. Cmin opcional de X, com respeito a Y, no contexto de R, é uma expressão da seguinte forma: Cmin-Opt[R(X/Y)] = H, onde H > 0 e impõe que se uma instância y de Y

está envolvida em R, então o número de instâncias diferentes de X relacionadas a y é maior que ou igual a H.

– Definição 3. Cmin de X, com respeito a Y, no contexto de R, é uma expressão da seguinte forma: Cmin[R(X/Y)] = H, onde H > 0 e impõe que o número de instâncias diferentes de X relacionadas a qualquer y de Y por meio de R é maior que ou igual a H.

Outros dois tipos significativos de restrições: i) dependências funcionais e ii) restrições de existência; que são frequentemente usados na modelagem conceitual, também podem ser expressos através de RCs.

CAPÍTULO IV – O PROJETO UNIVERSIDATA

O trabalho aqui apresentado está inserido num projeto mais amplo, o projeto UniversiData [SILV 1990], desenvolvido no Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro. Este capítulo visa descrever os objetivos e características de tal projeto.

IV.l – ORIGENS E PROPOSTAS

O projeto UniversiData surgiu como uma proposta de integrar as informações ao gerar um banco de dados único dos diversos sistemas de informação da UFRJ. Para isso, busca-se uma forma de controle e uniformização das bases de dados utilizadas pelos diversos sistemas. Portanto, o ambiente é orientado para os dados e não para os aspectos funcionais destes sistemas, ou seja, a idéia é obter a integração através da utilização de bases de dados compartilhadas. As principais vantagens de tal procedimento são a padronização, a diminuição de redundância entre os diversos sistemas e, conseqüentemente, a diminuição de inconsistências.

Outro aspecto importante do projeto é a homogeneização do ambiente, tanto para a definição como para a manipulação do banco, de forma que haja uma uniformização do conhecimento sobre a base de dados para usuários, analistas e programadores. Para isso, o Modelo Entidade-Relacionamento é utilizado nas fases de modelagem conceitual e implementação.

Considerando os fatores acima, surgiu, então, a idéia do desenvolvimento de um ambiente que permitisse a construção desse banco de dados único com características favoráveis :

a) ao bom entendimento do mesmo por parte do usuário;

b) ao processo de manutenção dos dados;

c) à facilidade de se executar tarefas simples de acesso ao banco de dados tais como produção de relatórios e execução de consultas **ad hoc;**

d) a introdução de facilidades para utilização de técnicas de inteligência artificial para manipulação da base de dados; e

e) à manutenção do sistema o mais possível independente do **hardware** utilizado, mantendo uma **interface** completamente uniforme.

Este ambiente é denominado Quick-DB e é composto por um conjunto de ferramentas de **software** que o tornam prático, simples e didático no desenvolvimento de aplicações de bancos de dados. O Quick-DB é um ambiente para projeto, manutenção e manipulação de bancos de dados construídos sobre o Modelo Entidade/Relacionamento.

Quick-DB está baseado na idéia de um ambiente uniforme, gráfico, onde todos os componentes referem-se a uma extensão do Modelo E-R. Este enfoque permite que seus usuários sigam uma linha de trabalho consistente, simplificando a metodologia e evitando a introdução de conceitos estranhos ao modelo básico [SILV 19881.

IV.2 – FACILIDADES DO QUICK-DB

Pela utilização do QUICK-DB é possível [SILV 1992]:

. Definir diagramas do tipo Entidade-Relacionamento completos, incluindo extensões como entidades associativas, acessso ordenado e generalização;

. Verificar diagramas;

. Construir automaticamente um dicionário de dados durante a especificação do diagrama;

. Gerar descrições de bancos de dados equivalentes aos diagramas para implementação em sistemas de grande porte;

. Gerar bibliotecas de objetos e métodos para acesso aos bancos de dados descritos nos diagramas;

. Definir sistemas de regras de produção para a construção de sistemas especialistas, fortemente acoplados aos bancos de dados;

. Definir atributos virtuais através de sistemas de cláusulas simples;

. Folhear os bancos de dados através de uma **interface** gráfica;

. Gerar aplicações simples para manutenção dos bancos de dados.

IV.3 – OS COMPONENTES DO QUICK-DB

O QUICK-DB é formado por vários componentes interligados como descrito na figura 9. Estes componentes representam módulos do sistema que formam um conjunto de ferramentas de **software** cujas **interfaces** são similares e amigáveis e que caracterizam um ambiente integrado e homogêneo.

– **Diagramador E-R** : consiste de uma ferramenta gráfica para especificação de diagramas E-R, orientado por meio de cardápios gráficos. Durante essa fase, o usuário deve definir os elementos do Modelo E-R tais como entidades, relacionamentos, atributos, ordens de acesso, chaves, generalizações e cardinalidade dos relacionamentos. As demais ferramentas tomam o diagrama aqui definido como base, estabelecendo a unidade visual que o sistema oferece.

– **Dicionário de Dados :** este componente é totalmente orientado ao Modelo E-R e tem como característica principal sua plena capacidade de representar todos os elementos do Modelo E-R, como por exemplo a herança de atributos por parte de atributos que participam de uma generalização. O dicionário de dados vai sendo gerado à medida que o diagrama estiver sendo definido.

Após a criação do diagrama E-R, segue a fase de geração de uma descrição da base de dados, em micro ou **mainframe.**

– **Gerador de Bancos de Dados :** caso a escolha seja em **mainframe,** este módulo é utilizado para, a partir do dicionário de dados, gerar as descrições das bases de dados para o SGBD correntemente em uso. Há um protótipo construído para esta etapa responsável por gerar a descrição de um BD para ser usado remotamente [TOSO 19901. Neste protótipo, utiliza-se a linguagem de definição de dados desenvolvida pela UNISYS chamada DASDL que faz parte do **software** DMS II [UNIS 19841.

– **Servidor de Bancos de Dados :** Após a geração do banco objeto pelo módulo acima, todos os acessos ao banco via diagrama E-R serão feitos através do módulo Servidor de Bancos de Dados. Este módulo traduz as operações realizadas sobre o diagrama em operações sobre o banco objeto, além de coletar os resultados de sua execução e enviá-los aos módulos solicitantes. Desta forma, obtém-se um grau de independência em relação aos programas de aplicação, já que estes estão baseados em uma visão E-R do banco de dados. Caso haja uma mudança no SGBD utilizado, basta trocar o servidor.

– **Gerador de Rotinas de Manipulação :** caso a opção seja a geração de um banco de dados em micro, então o Gerador de Rotinas de Manipulação [DELV 1991] é utilizado. Este módulo é responsável por gerar a descrição deste BD local juntamente com um conjunto de métodos capazes de permitir a manipulação dos dados desse BD autônomo. As rotinas de manipulação usam o conceito de **surrogate** [CODD 1979], que permite obter uma identificação única para as instâncias de todos os objetos definidos no diagrama. O **surrogate** é controlado pelas rotinas de manipulação e é transparente aos usuários das rotinas. O mapeamento entre o Modelo Entidade-Relacionamento e estas rotinas de manipulação tem como base o mapeamento entre o Modelo Entidade-Relacionamento e o Modelo Relaciona1 [KORT 1986]. A idéia básica é representar os componentes do modelo E-R em forma de relações.

– **Módulo de Consultas :** o ambiente de **interface** gráfica é explorado para que se ofereça ao usuário casual a possibilidade de especificar consultas de modo simples e intuitivo. A idéia é construir a expressão de consulta, isto é, sua especificação, num processo estreitamente ligado ao diagrama E-R e dirigido pelo sistema. O usuário pode consultar, por meio de **interface** gráfica, não só bancos de dados locais como também remotos, de grande porte, com o auxílio de servidores [TRIN 1991].

– **Gerador de Relatórios :** tem como objetivo diminuir a dependência dos programas de aplicação em relação as bases de dados, gerando relatórios de baixa e média complexidade de uma forma total ou semi-automatizada.

– **Shell para Sistemas Especialistas :** baseada no paradigma Objeto-Atributo-Valor para representação do conhecimento. Neste módulo foi definida uma linguagem para construção de regras que abriga elementos como termos e quantificadores, que são acoplados ao servidor de bancos de dados de forma natural e não ambígua. Regras passam a ser uma alternativa adicional para consulta e manipulação do banco de dados [PACI 19911.

– **Gerador de Aplicações :** Este componente tem como objetivo a geração de aplicações simples para manutenção das bases de dados. Dessa forma, a construção de sistemas estará se beneficiando das funções já definidas no ambiente oferecido pelo Quick-DB. O módulo Gerador de Aplicações é justamente o objeto de estudo desta tese.

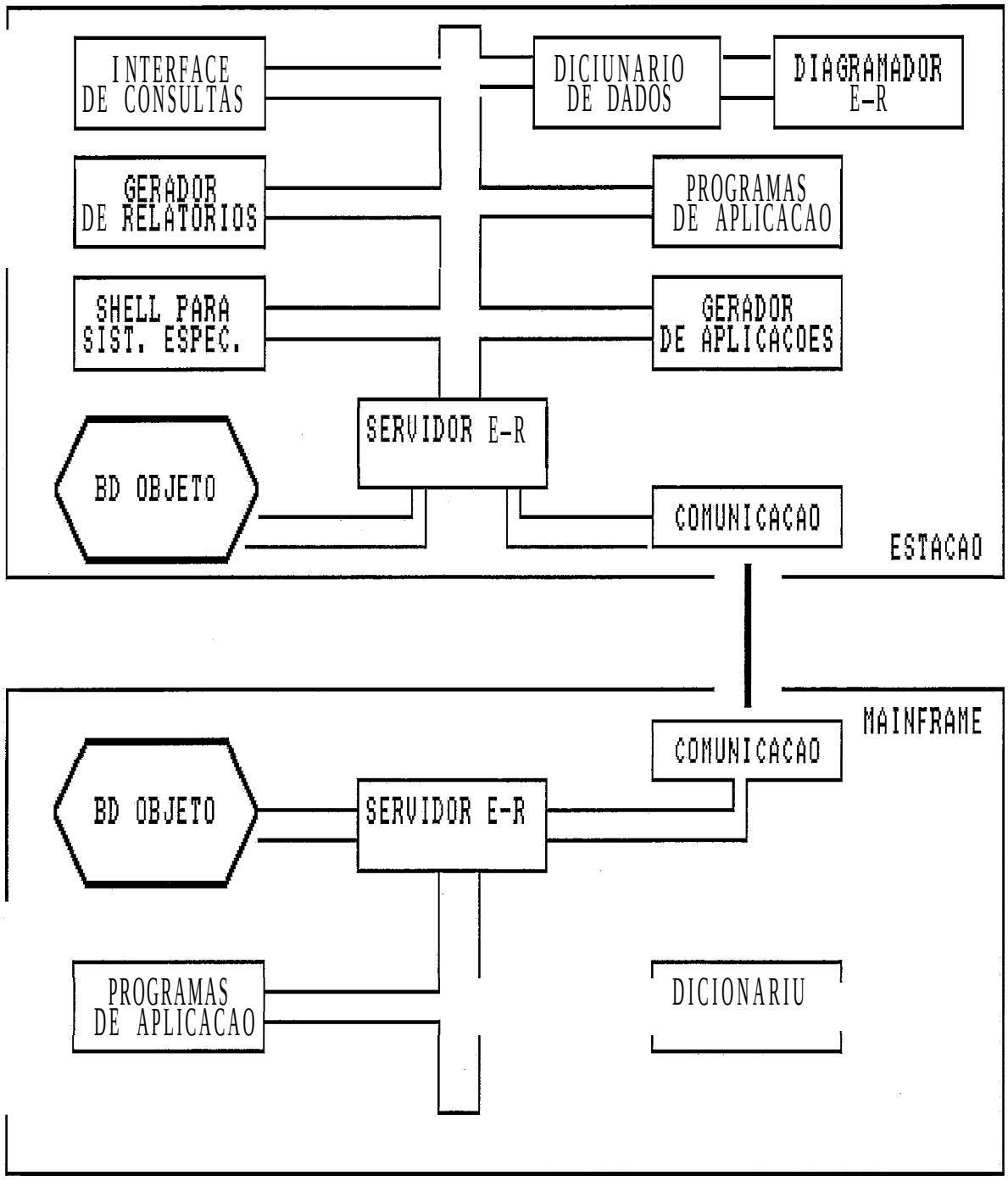


FIGURA 9 - QUICK-DB

IV.4 – CONCLUSÃO

Uma consideração importante se refere a **interface** entre os diversos módulos e o servidor de banco de dados. O servidor localiza-se no **mainframe** ou no micro para o qual foi gerado. A **interface** com este pode ser local ou através de uma ligação remota. Isto permite que os demais módulos possam aproveitar as facilidades gráficas de equipamentos de pequeno porte juntamente com as potencialidades e capacidade de processamento do **mainframe.** Para tal, foi feita uma ligação entre a estação remota e o **mainframe,** atendida por servidores, que se utilizam de primitivas de comunicação implementadas por um protocolo desenvolvido especialmente para este fim.

O projeto UniversiData proporciona um ambiente integrado para projeto de bancos de dados e desenvolvimento de sistemas aplicativos que utilizam bases de dados compartilhadas, definidas de uma maneira uniforme. Além da integração dos sistemas assim desenvolvidos, obtém-se um grau de independência em relação aos programas de aplicação e ao SGBD utilizado.

CAPÍTULO V – A SOLUÇÃO PROPOSTA

V.1 – INTRODUÇÃO

Nos capítulos anteriores descrevemos os assuntos mais relevantes à evolução deste trabalho, partindo do que hoje existe em termos de tecnologias de reutilização e a partir da teoria de modelagem de dados. Mostrou-se, ainda, uma descrição do ambiente no qual se insere este trabalho e os paradigmas nele adotados. Agora, iremos definir o Módulo Gerador de Aplicações, suas características, seus conceitos e a teoria utilizada.

Dentre as tecnologias conhecidas para a reutilização de **software,** o gerador de aplicação destaca-se pelo grande benefício proporcionado. Os geradores de aplicação se originaram a partir dos geradores de relatórios, como o RPG da IBM. Posteriormente, foram sendo feitas extensões para permitir a manipulação de dados, o gerenciamento do banco de dados e a geração de relatórios mais sofisticados.

Os padrões de arquitetura usados na geração são determinados pela identificação de uma família genérica de programas com grandes oportunidades de parametrização. Os valores dos parâmetros podem ser especificados pelo usuário através de um diálogo mantido com o sistema [BURN 19881.

V.2 – DEFINIÇÃO DAS CARACTERÍSTICAS

Inicialmente, é importante ressaltar que o projeto UniversiData opta por uma visão calcada nos dados em detrimento dos aspectos puramente funcionais do sistema. Deve ser considerado que já existe um modelo conceitual dos dados do sistema e que a base de dados deve ser aproveitada. O gerador de aplicações aqui proposto procura adotar técnicas de geração voltadas para um ambiente de administração de dados.

Como já foi visto anteriormente, o projeto UniversiData apresenta uma ferramenta gráfica, chamada de diagramador E-R, para especificação de diagramas E-R e as demais ferramentas do projeto devem tomar o diagrama definido pelo diagramador E-R como base.

O dicionário de dados pode ser considerado como um banco de dados que armazena informações sobre o esquema que foi especificado pelo diagramador E-R. Assim, o módulo Gerador de Aplicações recebe como entrada o dicionário de dados (com as definições de entidades e relacionamentos, seus atributos, suas chaves e ordens, as cardinalidades dos relacionamentos, etc) e

gera automaticamente um conjunto de telas e transações com todas as funções para criação e manutenção da base de dados de uma aplicação.

Para cada entidade ou relacionamento, são geradas as transações de alteração, exclusão e inclusão de instâncias. Cada transação possui uma ou mais telas de manutenção para interface com o usuário. As transações realizam algumas consistências automaticamente e erros detectados durante o processamento da transação são indicados e a transação não se realiza.

V.3 – RESTRIÇÕES DE INTEGRIDADE NAS TRANSAÇÕES GERADAS

Todas as transações de alteração, exclusão e inclusão são implementadas de maneira a impedir que o usuário viole determinadas regras de integridade. A idéia básica é que o banco de dados deve estar obrigatoriamente íntegro após a realização de uma transação. Assim, pode-se adotar dois procedimentos para evitar que o banco esteja em um estado incorreto após uma transação: a) rejeitar a transação que deixe o banco de dados em um estado incorreto e b) realizar operações adicionais para garantir que o resultado final seja um estado correto.

Antes de detalhar estas regras, é importante fazer algumas observações que devem ser consideradas na hora de se especificar o esquema E-R. Estas observações estão ilustradas na figura 10.

É possível definir uma ou mais chaves para cada entidade ou relacionamento. Toda entidade deve possuir ao menos uma chave. Os relacionamentos podem não possuir chaves próprias, nesse caso, para se referenciar a instância de um relacionamento, devem ser usadas as chaves das entidades envolvidas.

As chaves das entidades podem possuir um ou mais atributos. Estes atributos podem pertencer a própria entidade ou ser herdados de seus supertipos.

As chaves próprias dos relacionamentos podem possuir um ou mais atributos. Estes atributos podem pertencer ao próprio relacionamento, às entidades envolvidas ou ainda ser herdados dos supertipos das entidades envolvidas.

Os relacionamentos também possuem as chaves indiretas, que são as chaves das entidades envolvidas. Assim, para se referenciar a instância de um relacionamento, podem ser utilizados : a) alguma chave própria do relacionamento, b) alguma chave das entidades envolvidas e c) combinação de chaves das entidades envolvidas, desde que não haja na combinação mais de uma chave pertencente à mesma entidade.

ENTIDADE	RELACIONAMENTO	
CHAVE PROPRIA OBRIGATORIA	CHAVE PROPRIA OPCIONAL	CHAVE INDIRETA OBRIGATORIA
atributos da chave	atributos da chave	atributos da chave
pertencem a propria entidade **OU** sao herdados dos supertipos das propria entidade	pertencem ao proprio relacionamento **OU** pertencem as entidades envolvidas **OU** sao herdados dos supertipos das entidades envolvidas	pertencem as entidades envolvidas **OU** sao herdados dos supertipos das entidades envolvidas

FIGURA 10 - QUADRO EXPLICATIVO DAS CHAVES

V.3.1 – INTEGRIDADE DE ENTIDADE

Todo atributo participante de chave não pode ter valor nulo. Ocorre aqui uma variação em relação à definição encontrada em [DATE 19861, que se refere somente a chave primária. Como consequência desta restrição, ocorre o seguinte:

. Na inclusão ou alteração de uma instância de entidade, nenhum atributo que pertença a alguma chave desta entidade ou pertença a alguma chave própria de relacionamento, poderá ficar com o valor nulo.

. Na inclusão ou alteração de uma instância de relacionamento, nenhum atributo que pertença a alguma chave própria do relacionamento, poderá ficar com o valor nulo.

V.3.2 – UNICIDADE DE CHAVE

Pela própria definição de chave, que é o conjunto de atributos que identifica unicamente uma instância de uma entidade ou relacionamento, teremos como consequência o seguinte:

. Na inclusão ou alteração de uma instância de entidade ou de relacionamento, não poderá haver instâncias com chaves iguais.

V.3.3 – INTEGRIDADE REFERENCIAL

Se uma relação R2 possui uma chave estrangeira CE correspondendo à chave primária CP de alguma relação R1, então todo valor de CE em R2 deve ser igual ao valor de CP em alguma instância de R1. Ocorre aqui uma variação em relação à definição encontrada em [DATE 19861, que afirma que todo valor de CE em R2 deve ser igual ao valor de CP em alguma instância de R1 **ou deve ser nulo.**

No nosso caso, a chave primária CP de alguma relação R1 se refere ao **surrogate** da instância de uma entidade e a chave estrangeira CE em R2 se refere aos **surrogates** das instâncias de entidades envolvidas em um relacionamento. Como conseqüência desta restrição, ocorre o seguinte:

a) Na inclusão de uma instância de relacionamento, as instâncias das entidades envolvidas já devem existir. Caso ainda não existam, há duas opções:

al) a transação se propaga, ou seja, as instâncias inexistentes das entidades envolvidas são incluídas, para depois se incluir a instância do relacionamento.

a2) a transação é cancelada, ou seja, a inclusão da instância do relacionamento não se realiza.

b) Na exclusão de uma instância de entidade, pode acontecer que haja instâncias de relacionamento que lhe correspondam. Nesse caso, há duas opções:

bl) a transação se propaga, ou seja, as instâncias de relacionamento que lhe correspondem são excluídas, para depois se excluir a instância da entidade.

b2) a transação é cancelada, ou seja, a exclusão da instância da entidade não se realiza.

V.3.4 – RESTRIÇÕES DE CARDINALIDADE RELATIVAS

Já foi vista a definição de [LENZ 1983] para restrições de cardinalidade relativas, segundo a qual Cmin e Cmax de um conjunto de entidades em relação a um outro conjunto de entidades, no contexto de um relacionamento, representam o

número mínimo e máximo de instâncias de um conjunto de entidades que podem se relacionar a uma instância de um outro conjunto de entidades através de um determinado relacionamento. Como conseqüência desta restrição, ocorre o seguinte:

a) Na exclusão de uma instância de entidade El, pode ocorrer propagação se houver instâncias do relacionamento R que lhe correspondam. Nesse caso, se Cmin [R(E1/E2)] = 1, isso significa que toda instância da entidade E2 deve estar relacionada a pelo menos uma instância da entidade E1 através do relacionamento R. Como as instâncias do relacionamento R devem ser excluídas, pode acontecer que haja instâncias da entidade E2 que não estão mais relacionadas a pelo menos uma instância da entidade El. Nesse caso, há duas opções:

al) a transação se propaga, isto é, estas instâncias da entidade E2 também são excluídas e deve se verificar se estas novas exclusões vão acarretar novas propagações de exclusão ou não.

a2) a transação é cancelada, isto é, a exclusão da instância da entidade E1 e as exclusões das instâncias do relacionamento R que lhe correspondem não são realizadas.

b) Na exclusão de uma instância do relacionamento R, se Cmin [R(E1/E2)] = 1, isso significa que toda instância da entidade E2 deve estar relacionada a pelo menos uma instância da entidade E1 através do relacionamento R. A exclusão da instância do relacionamento R pode fazer com que haja alguma instância da entidade E2 que não está mais relacionada a pelo menos uma instância da entidade El. Nesse caso, há duas opções:

b1) a transação se propaga, isto é, esta instância da entidade E2 também é excluída e deve se verificar se esta nova exclusão vai acarretar novas propagações de exclusão ou não.

b2) a transação é cancelada, isto é, a exclusão da instância do relacionamento R não é realizada.

c) Na inclusão de uma instância da entidade El, se Cmin [R(E2/E1)] = 1, isso significa que toda instância da entidade E1 deve estar relacionada a pelo menos uma instância da entidade E2 através do relacionamento R. Nesse caso, há duas opções:

c1) a transação é propagada, isto é, é necessário informar a instância da entidade E2 a qual a instância da entidade E1 vai se relacionar. A instância da entidade E2 pode já existir ou pode ser necessário que se tenha que incluí-la. Há então duas novas opções:

c11) a instância da entidade E2 já existe. Nesse caso, se Cmax [R(E1/E2)] = 1 e a instância da entidade E2 já estiver relacionada a uma outra instância da entidade E1 através de R, a transação é rejeitada e a inclusão da instância da entidade E1 não é realizada. Caso Cmax [R(E1/E2)] diferente de 1 ou se a instância da entidade E2 não estiver relacionada a uma outra instância da entidade E1 através de R, é realizada a inclusão da instância da entidade E1 e a inclusão da instância do relacionamento R.

c12) a instância da entidade E2 não existe. Nesse caso, é realizada a inclusão da instância da entidade E2, a inclusão da instância da entidade E1 e a inclusão da instância do relacionamento R. A inclusão da instância da entidade E2 pode acarretar uma nova propagação de inclusão.

c2) a transação é cancelada, isto é, a inclusão da instância da entidade E1 não se realiza.

d) Na inclusão de uma instância do relacionamento R, é necessário informar as instâncias das entidades E1 e E2 que vão estar relacionadas. Estas instâncias das entidades E1 e E2 podem já existir ou pode ser necessário que se tenha que incluí-las. Há então quatro opções:

d1) a instância da entidade E1 já existe e a instância da entidade E2 já existe. Nesse caso, se Cmax [R(E1/E2)] = 1 e a instância da entidade E2 já estiver relacionada a uma outra instância da entidade E1 através de R, a transação é rejeitada e a inclusão da instância do relacionamento R não é realizada. Da mesma forma, se Cmax [R(E2/E1)] = 1 e a instância da entidade E1 já estiver relacionada a uma outra instância da entidade E2 através de R, a transação é rejeitada e a inclusão da instância do relacionamento R não é realizada. Caso as hipóteses acima não aconteçam, é realizada a inclusão da instância do relacionamento R.

d2) a instância da entidade E1 já existe e a instância da entidade E2 não existe. Nesse caso, se Cmax [R(E2/E1)] = 1 e a instância da entidade E1 já estiver relacionada a uma outra instância da entidade E2 através de R, a transação é rejeitada e a inclusão da instância do relacionamento R não

é realizada. Caso a hipótese acima não tenha acontecido, é realizada a inclusão da instância da entidade E2 e a inclusão da instância do relacionamento R. A inclusão da instância da entidade E2 pode acarretar uma nova propagação de inclusão.

d3) a instância da entidade E2 já existe e a instância da entidade E1 não existe. Nesse caso, se Cmax [R(E1/E2)] = 1 e a instância da entidade E2 já estiver relacionada a uma outra instância da entidade E1 através de R, a transação é rejeitada e a inclusão da instância do relacionamento R não é realizada. Caso a hipótese acima não tenha acontecido, é realizada a inclusão da instância da entidade E1 e a inclusão da instância do relacionamento R. A inclusão da instância da entidade E1 pode acarretar uma nova propagação de inclusão.

d4) a instância da entidade E1 não existe e a instância da entidade E2 não existe. Nesse caso, é realizada a inclusão da instância da entidade E2, a inclusão da instância da entidade E1 e a inclusão da instância do relacionamento R. A inclusão da instância da entidade E2 e a inclusão da instância da entidade E1 podem acarretar novas propagações de inclusão.

V.4 - CONCEITO DE MACROTRANSAÇÃO E MICROTRANSAÇÃO

Como foi visto no item anterior, as transações são geradas considerando que o banco de dados deve estar íntegro após a realização de uma transação. Foi visto também que existem determinadas ações que deixam o banco em um estado incorreto e que, nesse caso, dois procedimentos podem ser adotados. O segundo procedimento, que é realizar operações adicionais para garantir que o resultado final seja um estado correto, vai dar origem aos conceitos de macrotransação e microtransação, que serão explicados a seguir:

Denomina-se microtransação qualquer alteração, exclusão ou inclusão realizada em uma instância de entidade ou relacionamento considerada de forma isolada. É importante ressalvar que uma microtransação pode deixar o banco de dados em um estado incorreto.

A inclusão de uma instância de entidade é um exemplo de microtransação. A exclusão de uma instância de relacionamento também é uma microtransação.

Uma macrotransação pode ser composta de uma ou mais microtransações. Denomina-se macrotransação ao conjunto de

microtransações que são realizadas em conjunto até deixar o banco de dados em um estado íntegro.

Para exemplificar o conceito de macrotransação podemos citar o seguinte exemplo : a microtransação de exclusão de uma instância de entidade faz com que seja necessário também executar as microtransações de exclusão das instâncias do relacionamento que lhe correspondem; o conjunto de todas estas microtransações de exclusão denomina-se macrotransação.

Nos casos das transações de alteração, a macrotransação é sempre composta de somente uma microtransação, que é a própria alteração da instância, visto que a alteração não acarreta propagação.

V.5 - A CONSTRUÇÃO DO GERADOR DE TRANSAÇÕES

Nesta seção, será mostrado como o Gerador de Transações se enquadra nos conceitos teóricos sobre construção de geradores de aplicação analisados anteriormente neste trabalho.

V.5.1 - ARQUITETURA DO GERADOR DE TRANSAÇÕES

[BURN 1986] descreve o módulo de diálogo, o módulo intermediário e o módulo gerador de código; que são as estruturas necessárias para quase todos os geradores. É feita a ressalva de que em algumas aplicações, o módulo intermediário pode ser omitido.

O módulo de diálogo do Gerador de Transações é o diagramador E-R. O usuário interage com o diagramador E-R para obter uma especificação completa e consistente do problema, que será a especificação do programa a ser gerado. Quando o diálogo estiver completo, a especificação do programa também estará completa e será armazenada no dicionário de dados.

O módulo intermediário do Gerador de Transações é o Gerador de Rotinas de Manipulação [DELV 1991]. O meta-programa resultante desta etapa é o conjunto de rotinas de manipulação, que são escritas na linguagem de alto nível TURBO PASCAL 6.0 [PASC 1990]. O meta-programa descreve de forma sucinta as operações que devem ser realizadas pela aplicação gerada; no nosso caso, as rotinas de manipulação descrevem os detalhes internos das transações, de maneira a permitir que a aplicação manipule os dados sem que tenha conhecimento do nível físico dos dados.

O módulo gerador de código do Gerador de Transações é o próprio gerador proposto neste trabalho. O produto resultante desta etapa é a aplicação com as transações de alteração, exclusão e inclusão. A aplicação é gerada a partir da especificação gerada pelo módulo de diálogo e utiliza os métodos criados pelo módulo intermediário.

V.5.2 – ABORDAGEM DE CONSTRUÇÃO DO GERADOR DE TRANSAÇÕES

Segundo a metódica abordagem encontrada em [CLEA 1988] e já detalhada anteriormente, podemos considerar que o Gerador de Transações foi construído seguindo-se os seguintes passos:

Primeiro, o reconhecimento de domínios. Foi analisado onde um gerador de aplicações poderia ser utilizado no contexto do projeto UniversiData. O reconhecimento da existência de padrões na execução das transações foi o caminho básico para concluir que estas transações seriam um domínio potencial para um gerador de transações.

Segundo, a definição de limites do domínio, isto e, a determinação do alcance do gerador. Resolvemos não incluir as operações de consulta devido ao fato de já existir no projeto um módulo de consultas. Concluímos que um módulo de geração de relatórios, embora importante, deveria ser o assunto de algum projeto final de curso de graduação inserido no projeto. Alguns aspectos funcionais também foram considerados e decidimos que seriam uma interessante extensão deste trabalho, mas que ficariam fora do escopo desta tese. Com isso, o alcance definido para o Gerador de Transações proposto neste trabalho foi o seguinte: as transações de alteração, exclusão e inclusão.

Terceiro, a definição de um modelo matemático como base. A recomendação de que se tenha um modelo matemático como base para um gerador de aplicações foi satisfeita com a análise da formalização matemática do modelo E-R proposta em [SILV 1989].

Quarto, a definição das partes variáveis e invariáveis. Observamos que as partes variáveis são as partes contidas na especificação, ou seja, no esquema E-R definido pelo usuário. Podemos relacionar as partes variáveis com **o que a aplicação deve fazer.** As partes invariáveis são as partes comuns de quaisquer transações. Podemos relacionar as partes invariáveis com **o como a aplicação deve fazer.** Estas partes invariáveis foram colocadas dentro da lógica do próprio Gerador de Transações.

Quinto, a definição da especificação de entrada. Como **já** foi enfatizado anteriormente, o diagrama definido pelo diagramador E-R deve servir como base para as demais ferramentas do projeto

UniversiData. Por isso, o formato escolhido para a especificação de entrada para o gerador foi o diagrama E-R.

Sexto, a definição dos produtos. O produto do gerador aqui proposto é um programa para a realização das transações de atualização do banco de dados. Outros produtos alternativos seriam dados de teste, documentação, relatórios, etc.

Por fim, a implementação do gerador. Foi construído um programa que recebe a especificação fornecida pelo usuário e gera o produto desejado , isto é, gera o programa para a realização das transações de atualização.

V.5.3 – COMPONENTES BÁSICOS DO GERADOR DE TRANSAÇÕES

De acordo com o estudo feito por [HORO 19851, podemos considerar que o Gerador de Transações possui os seguintes componentes básicos:

. **SGBD.** O Gerador de Transações possui o seu próprio SGBD. Para o controle do nível físico deste SGBD, utiliza-se o TURBO PASCAL DATABASE TOOLBOX. O banco de dados geralmente possui dois tipos de arquivos: a) um dicionário de dados, que no nosso caso armazena as informações sobre o esquema E-R e b) os arquivos de dados que contêm os dados que serão atualizados pelas transações geradas.

. **Gerador de relatório.** Em sua atual fase, o gerador de aplicações proposto neste trabalho, que na realidade é somente um gerador de transações, não apresenta o componente gerador de relatório. Da mesma forma, não possui o componente pacote gráfico, que nada mais é do que um gerador de relatório gráfico.

. **Procedimentos de manipulação.** É o cerne do Gerador de Transações aqui proposto. O Gerador de Transações permite a inclusão de registros, a exclusão de registros, a alteração de dados dos registros e verifica restrições de integridade. Os procedimentos de manipulação que o gerador não realiza são a verificação de restrições de segurança e o acesso e listagem de registros.

V.6 – ESPECIFICAÇÃO DAS TRANSAÇÕES DE ATUALIZAÇÃO

Os fundamentos lançados para a solução proposta neste trabalho são concretizados na ferramenta denominada Gerador de Transações. São implementados seis tipos de transação: alteração de entidade, alteração de relacionamento, exclusão de entidade,

exclusão de relacionamento, inclusão de entidade e inclusão de relacionamento. Faremos nesta seção uma breve descrição das etapas para se realizar cada uma destas transações, que serão exemplificadas no apêndice.

V.6.1 – ALTERAÇÃO DE ENTIDADE

Vide transação 3 do apêndice.

1) Identificação da chave de acesso da entidade. Vide tela 3.2 do apêndice. Todas as chaves da entidade e todas as chaves herdadas pela entidades são exibidas ao usuário para que este possa escolher qual a chave que será usada para identificar a instância da entidade. A transação prossegue na etapa 2. Se houver apenas uma chave de acesso para esta entidade, a transação de alteração de entidade tem início na etapa 2.

2) Entrada dos valores dos atributos da chave. Vide tela 3.3 do apêndice. O usuário preenche os campos referentes aos atributos da chave. Se existir uma instância desta entidade com esse conteúdo de chave, a transação continua na etapa 3. Caso contrário, aparece mensagem de erro.

3) Alteração da instância da entidade. Vide tela **3.4** do apêndice. Os valores atuais dos atributos da instância identificada são exibidos ao usuário e este preenche os campos que desejar alterar. Se a alteração feita pelo usuário resultar em chave duplicada, aparece mensagem de erro. Caso contrário, a microtransação de alteração teve sucesso.

V.6.2 – ALTERAÇÃO DE RELACIONAMENTO

Vide transação 7 do apêndice.

1) Identificação da chave de acesso do relacionamento. Vide tela 7.1 do apêndice. Todas as chaves próprias do relacionamento, todas as chaves próprias e herdadas das entidades envolvidas; e combinações entre chaves próprias e herdadas de uma entidade envolvida com chaves próprias e herdadas da outra entidade envolvida são exibidas ao usuário para que este possa escolher qual a chave que será usada para identificar a instância do relacionamento a ser alterado. Aparecem aqui duas alternativas: a) o usuário escolhe uma chave própria do relacionamento e a transação prossegue na etapa 2.1 ou b) o usuário escolhe uma chave de entidade envolvida ou uma combinação entre estas e a transação prossegue na etapa 2.2.

2.1) Entrada dos valores dos atributos da chave própria do relacionamento. Vide tela 7.2 do apêndice. O usuário preenche os campos referentes aos atributos da chave. Se existir uma instância deste relacionamento com esse conteúdo de chave, a transação continua na etapa 4. Caso contrário, aparece mensagem de erro.

2.2) Entrada dos valores dos atributos da chave própria ou herdada da entidade envolvida ou entrada dos valores dos atributos da combinação de chaves próprias ou herdadas das entidades envolvidas. Vide tela 8.2 do apêndice. O usuário preenche os campos referente(s) aos atributos da(s) chave(s). A transação prossegue na etapa 3.

3) Confirmação da instância do relacionamento a ser alterada. Vide tela 8.3 do apêndice. As instâncias do relacionamento que correspondem aos valores da(s) chave(s) fornecidas pelo usuário são exibidas (os atributos da instância são exibidos) uma a uma, até que o usuário confirme qual das instâncias vai ser alterada e a transação prossegue na etapa 4. Caso nenhuma instância seja confirmada, aparece mensagem de erro.

4) Alteração da instância do relacionamento. Vide tela 7.3 do apêndice. Os valores atuais dos atributos da instância identificada são exibidos ao usuário e este preenche os campos que desejar alterar. Se a alteração feita pelo usuário resultar em chave duplicada, aparece mensagem de erro. Caso contrário, a microtransação de alteração teve sucesso.

V.6.3 - EXCLUSÃO DE ENTIDADE

Vide transações 4 e 5 do apêndice.

1) Vide etapa 1 de ALTERAÇÃO DE ENTIDADE. Vide tela 5.1 do apêndice.

2) Entrada dos valores dos atributos da chave. Vide telas 4.2 e 5.2 do apêndice. O usuário preenche os campos referentes aos atributos da chave. Se não existir uma instância desta entidade com esse conteúdo de chave, aparece mensagem de erro. Caso contrário, podem ocorrer duas alternativas: a) a exclusão da instância desta entidade não acarreta propagações de exclusão e a transação prossegue na etapa 3.1 ou b) a exclusão da instância desta entidade acarreta propagações de exclusão (pelo menos mais uma outra instância também deve ser excluída) e a transação prossegue na etapa 3.2.

3.1) Exibição da instância da entidade a ser excluída. Vide tela 4.3 do apêndice. Os atributos da instância da entidade a ser excluída são exibidos. A transação prossegue na etapa 3.1.1.

3.1.1) Confirmação da exclusão da instância da entidade. Vide tela 4.4 do apêndice. O usuário confirma se a instância vai ser realmente excluída. Caso seja confirmado, a microtransação de exclusão é efetivada.

3.2) Decisão de ver ou não as instâncias a ser excluídas. Vide tela 5.3 do apêndice. O usuário decide se quer ou não ver todas as instâncias que vão ser excluídas. Caso decida que quer ver, a transação prossegue na etapa 3.2.1. Caso decida que não, a transação prossegue na etapa 3.2.2.

3.2.1) Exibição de todas as instâncias que vão ser excluídas. Vide telas 5.4 e 5.5 do apêndice. As instâncias que vão ser excluídas são exibidas uma a uma (os seus atributos são exibidos). A transação prossegue na etapa 3.2.2.

3.2.2) Confirmação da exclusão das instâncias. Vide tela 5.6 do apêndice. O usuário confirma se todas estas instâncias vão ser realmente excluídas. Caso seja confirmado, a macrotransação de exclusão de todas estas instâncias é efetivada.

V.6.4 – EXCLUSÃO DE RELACIONAMENTO

Vide transação 8 do apêndice.

1) Vide etapa 1 de ALTERAÇÃO DE RELACIONAMENTO. Vide tela 8.1 do apêndice.

2.1) Entrada dos valores dos atributos da chave própria do relacionamento. Vide tela 7.2 do apêndice. O usuário preenche os campos referentes aos atributos da chave. Se não existir uma instância deste relacionamento com esse conteúdo de chave, aparece mensagem de erro. Caso contrário, podem ocorrer duas alternativas: a) a exclusão da instância deste relacionamento não acarreta propagações de exclusão e a transação prossegue na etapa 2.1.1 ou b) a exclusão da instância deste relacionamento acarreta propagações de exclusão (pelo menos mais uma outra instância também deve ser excluída) e a transação prossegue na etapa 2.1.2.

2.1.1) Exibição da instância do relacionamento a ser excluído. Vide tela 5.5 do apêndice. Os atributos da instância do relacionamento a ser excluído são exibidos. A transação prossegue na etapa 2.1.1.1.

2.1.1.1) Confirmação da exclusão da instância do relacionamento. O usuário confirma se a instância vai ser realmente excluída. Caso seja confirmado, a microtransação de exclusão é efetivada.

2.1.2) Vide etapa 3.2 de EXCLUSÃO DE ENTIDADE.

3.2.1) Vide etapa 3.2.1 de EXCLUSÃO DE ENTIDADE.

3.2.2) Vide etapa 3.2.2 de EXCLUSÃO DE ENTIDADE.

2.2) Entrada dos valores dos atributos da chave própria ou herdada da entidade envolvida ou entrada dos atributos da combinação de chaves próprias ou herdadas das entidades envolvidas. Vide tela 8.2 do apêndice. O usuário preenche os campos referente(s) aos atributos da(s) chave(s). A transação prossegue na etapa 2.2.1.

2.2.1) Confirmação da instância do relacionamento a ser excluído. Vide tela 8.3 do apêndice. As instâncias do relacionamento que correspondem aos valores da(s) chave(s) fornecidas pelo usuário são exibidas (os atributos da instância são exibidos) uma a uma, até que o usuário confirme qual das instâncias vai ser excluída. Caso nenhuma instância seja confirmada, aparece mensagem de erro, Caso contrário, podem ocorrer duas alternativas: a) a exclusão da instância deste relacionamento não acarreta propagações de exclusão e a transação prossegue na etapa 3a ou b) a exclusão da instância deste relacionamento acarreta propagações de exclusão (pelo menos mais uma outra instância também deve ser excluída) e a transação prossegue na etapa 2.1.2.

V.6.5 - INCLUSÃO DE ENTIDADE

Vide transações 1 e 2 do apêndice.

1) Inclusão da instância da entidade El. Vide telas 1.2 e 2.1 do apêndice. O usuário preenche os campos referentes aos atributos da entidade El. Se a inclusão feita resultar em chave duplicada, aparece mensagem de erro. Caso contrário, a microtransação de inclusão teve sucesso. Se a microtransação teve sucesso, podem ocorrer duas alternativas: a) a inclusão da instância da entidade E1 acarreta propagações de inclusão (pelo menos mais uma outra instância deve ser incluída) e a transação prossegue na etapa 2 ou b) a inclusão da instância da entidade E1 não acarreta propagações de inclusão e a macrotransação de inclusão termina com sucesso.

2) Decisão de propagar ou não a inclusão. Vide tela 2.3 do apêndice. O usuário decide se quer ou não continuar a macrotransação de inclusão. Caso decida que quer continuar, a transação prossegue na etapa 3. Caso decida que não quer continuar, a macrotransação de inclusão não se completa, isto é, a microtransação realizada na etapa 1 não se efetiva.

3) Confirmação da existência da instância da entidade Ej. Vide tela 2.4 do apêndice. Esta etapa vai acontecer para todas as entidades Ej com Cmin igual a 1 relacionadas a E1 através do relacionamento Rj. O usuário responde se a instância da entidade Ej que vai se relacionar com a instância da entidade E1 já foi incluída ou não. Se o usuário responde que a instância já foi incluída, a transação prossegue na etapa 4.1. Se o usuário responde que a instância ainda não foi incluída, a transação prossegue na etapa 4.2.

4.1) Identificação da chave de acesso da entidade Ej. Todas as chaves da entidade Ej e todas as chaves herdadas pela entidades Ej são exibidas ao usuário para que este possa escolher qual a chave que será usada para identificar a instância da entidade. A transação prossegue na etapa 4.1.1. Se houver apenas uma chave de acesso para esta entidade, não ocorre a etapa 4.1 e a transação vai direto para a etapa 4.1.1.

4.1.1) Entrada dos valores dos atributos da chave da entidade Ej. Vide tela 4.2 do apêndice. O usuário preenche os campos referentes aos atributos da chave. Se existir uma instância da entidade Ej com esse conteúdo de chave, a transação continua na etapa 4.1.2. Caso contrário, aparece mensagem de erro.

4.1.2) Inclusão da instância do relacionamento Rj entre E1 e Ej. Vide tela 2.5 do apêndice. O usuário preenche os campos referentes aos atributos do relacionamento Rj. Se a inclusão feita resultar em chave duplicada, aparece mensagem de erro. Caso contrário, a microtransação de inclusão do relacionamento Rj teve sucesso e a macrotransação de inclusão (composta da microtransação de inclusão de E1 e da microtransação de inclusão de Rj) termina com sucesso.

4.2) O processo se torna recursivo, com as etapas 1, 2, 3, 4.1, 4.1.1, 4.1.2 e 4.2 se repetindo. É importante observar que a entidade Ej passa a ser a nova entidade El. A etapa 3 vai acontecer para todas as entidades Ej com Cmin igual a 1 relacionadas a E1 através do relacionamento Rj (com exceção da entidade Ej que for a mesma entidade E1 anterior). Na etapa 4.1.2, a macrotransação de inclusão ainda não terminou, pois ainda faltar incluir o relacionamento Rj entre a entidade E1 original e a entidade Ej original. A transação prossegue na etapa 4.2.1.

4.2.1) Inclusão da instância do relacionamento Rj entre E1 original e Ej original. Vide tela 2.5 do apêndice. O usuário preenche os campos referentes aos atributos do relacionamento Rj. Se a inclusão feita resultar em chave duplicada, aparece mensagem de erro. Caso contrário, a microtransação de inclusão do relacionamento Rj teve sucesso e a macrotransação

de inclusão completa (com toda as propagações de inclusão) termina com sucesso.

V.6.6 – INCLUSÃO DE RELACIONAMENTO

Vide transação 6 do apêndice.

1) Confirmação da existência da instância da entidade E1 e da instância da entidade E2. Vide telas 6.1 e 6.2 do apêndice. As entidades E1 e E2 são as entidades envolvidas no relacionamento R em questão. As etapas **1.1, 1.1.1** e 1.2 vão acontecer duas vezes: uma vez para a entidade E1 e a segunda vez para a entidade E2. O usuário responde se a instância da entidade Ej já foi incluída ou não. Se o usuário responde que a instância já foi incluída, a transação prossegue na etapa **1.1.** Se o usuário responde que a instância ainda não foi incluída, a transação prossegue na etapa 1.2.

1.l) Identificação da chave de acesso da entidade Ej. Todas as chaves da entidade Ej e todas as chaves herdadas pela entidades Ej são exibidas ao usuário para que este possa escolher qual a chave que será usada para identificar a instância da entidade. A transação prossegue na etapa **1.1.1.** Se houver apenas uma chave de acesso para esta entidade, não ocorre etapa **1.1** e a transação vai direto para a etapa **1.1.1.**

1.1.1) Entrada dos valores dos atributos da chave da entidade Ej. Vide tela 4.2 do apêndice. O usuário preenche os campos referentes aos atributos da chave. Se existir uma instância da entidade Ej com esse conteúdo de chave, a transação continua na etapa 2. Caso contrário, aparece mensagem de erro.

1.2) Seguem as etapas da transação de inclusão de entidade já vistas anteriormente. A transação prossegue na etapa 2.

2) Inclusão da instância do relacionamento R entre E1 e E2. Vide tela 2.5 do apêndice. O usuário preenche os campos referentes aos atributos do relacionamento R. Se a inclusão feita resultar em chave duplicada, aparece mensagem de erro. Caso contrário, a microtransação de inclusão do relacionamento R teve sucesso e a macrotransação de inclusão completa (com todas as propagações de inclusão) termina com sucesso.

CAPÍTULO VI – IMPLEMENTAÇÃO

Nesse capítulo serão apresentadas as principais características físicas do módulo Gerador de Aplicações, incluindo aspectos estruturais e detalhes de implementação. Na primeira parte, serão abordadas algumas questões de Programação Orientada a Objetos, que é utilizada nas transações criadas pelo Gerador de Aplicações. Depois, é feita uma descrição mais detalhada do servidor E-R (Gerador de Rotinas de Manipulação) e dos objetos e métodos de acesso que ele cria e como estes são utilizados pelas transações. Por fim, são descritos detalhes da implementação do módulo Gerador de Transações, como o programa gerado está organizado e como o usuário realiza as transações através da **interface** TURBO VISION.

VI.l – ASPECTOS DE ORIENTAÇÃO A OBJETOS

Programação Orientada a Objetos é um método de programação que tenta modelar o mundo real da forma mais natural possível. Este estilo de programação possui maior estruturação, modularidade, abstração e capacidade de "esconder" informação [PASC 1989] em comparação com as diversas formas de programação estruturada utilizadas até agora.

Há três propriedades principais que caracterizam a POO :

a) **Encapsulamento:** O encapsulamento consiste na combinação de um registro convencional com os procedimentos e funções que o manipulam formando um novo tipo de dado, um objeto.

b) **Herança:** Consiste na definição de um determinado objeto e na sua reutilização para construir uma hierarquia de objetos descendentes, onde cada descendente herda o acesso ao código e aos dados de todos os seus ancestrais.

c) **Polimorfismo:** É uma facilidade oferecida na herança que permite a cada objeto a implementação de uma ação herdada da maneira que lhe for mais apropriada.

Um objeto possui uma memória interna em que valores podem ser armazenados e modificados ao longo da vida do mesmo. Possui também um comportamento, que consiste no conjunto de ações pré-definidas (métodos) através das quais o objeto responde à demanda de processamento por parte de outros objetos. Ao receber uma mensagem, o objeto seleciona a execução da ação que faz parte de seu comportamento. Após a execução, o objeto retorna o controle ao objeto que lhe enviou a mensagem. Segundo [MEYE

19881, é necessário distinguir entre objetos internos e externos. Objetos externos são a representação da realidade física cuja conduta tenta-se modelar na fase de projeto. Na fase de programação, a linguagem de programação não manipula esses objetos externos, mas sim com suas representações computacionais adequadas, os objetos internos.

Segundo [TAKA 1990], a POO apresenta as seguintes características:

a) **modularidade:** o conceito de objeto propicia uma alta modularidade na tarefa de projeto e programação;

b) **suporte a generalização/especialização:** permite a implementação direta do mecanismo de herança do modelo E-R;

c) **visão balanceada entre dados e processos:** possibilita uma visão integrada de dados e processos na modelagem conceitual de aplicações.

d) **desenvolvimento bottom-up de aplicações:** composição das aplicações a partir de objetos e métodos pré-existentes, que no caso são as rotinas de manipulação [DELV 1991];

e) **programação incremental:** a abordagem incremental de programação fica favorecida, visto que pequenas modificações podem ser facilmente efetuadas e testadas;

f) **reusabilidade:** a reutilização de código já existente fica muito facilitada.

Deste modo, observamos que a partir das entidades e relacionamentos definidas pelo usuário em seu esquema E-R, são gerados objetos na linguagem de programação TURBO PASCAL 6.0 [PASC 1990], cujos métodos são as rotinas de manipulação do banco de dados.

VI.2 – BANCO DE DADOS AUTÔNOMO

Esta seção descreve a organização interna de um banco de dados em microcomputador segundo a abordagem de orientação a objetos e os principais métodos que manipulam o mesmo.

VI.2.1 – SERVIDOR E-R

O acesso à base de dados é feito através do servidor E-R, visto na figura 11, que armazena uma coleção de métodos aplicáveis às classes de objetos que formam o banco de dados. As

transações geradas utilizam-se de bibliotecas de funções previamente definidas. Essas bibliotecas baseiam-se na descrição da base de dados conforme especificado através do diagramador E-R e do dicionário de dados. Para acessar um banco de dados local, construiu-se um utilitário que, a partir do esquema E-R especificado, ao ser executado, gera um servidor de banco de dados contendo as referidas bibliotecas [DELV 1991]. Esse servidor, desenvolvido para o TURBO PASCAL 6.0, possui **interface** orientada a objetos através dos métodos e manipula um banco de dados local autônomo. Os objetos, atributos e métodos são gerados de modo que os usuários possam ter acesso ao banco com facilidade. Dessa forma, o usuário tem a impressão de estar manipulando um banco de dados virtual, não se preocupando com informações de implementação do mesmo.

Além das transações criadas pelo Gerador de Aplicações utilizarem os métodos para acesso ao banco de dados, o servidor também traduz operações para a **interface** com sistemas especialistas e viabiliza as operações de consulta.

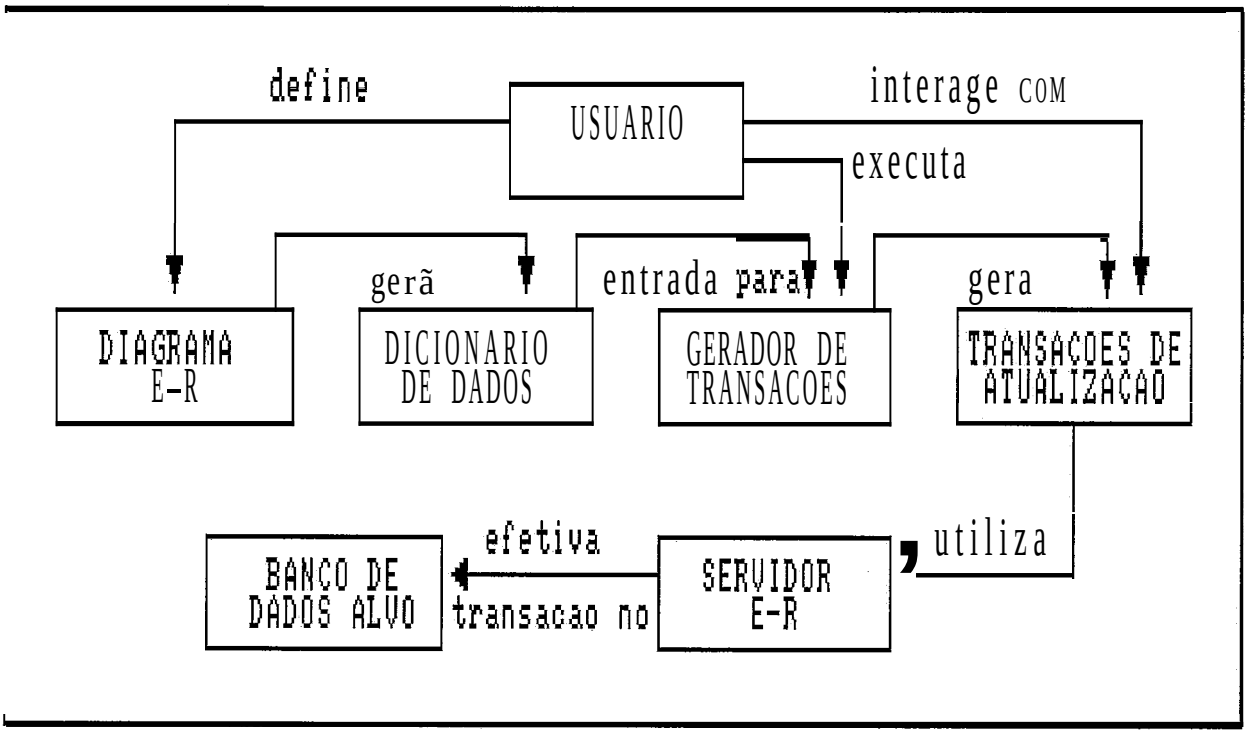


FIGURA 11 - ESQUEMA DO GERADOR DE TRANSACOES COM UTILIZACAO DO SERVIDOR

VI.2.2 – OBJETOS GERADOS

Conjuntos de entidades e relacionamentos são considerados uma mesma classe de objetos, identificada por T ENTIDADE. Cada entidade ou relacionamento especificada no esquema E-R gera um objeto T_ENTIDADE_<objeto>, que é sub-tipo de T_ENTIDADE e onde <objeto> é o nome da entidade ou relacionamento.

As ordens constituem uma classe de objetos que manipulam elementos dos conjuntos de entidades ou relacionamentos, identificada por T ORDEM. Cada ordem ou chave especificada no esquema E-R gera um–objeto T_ORDEM <objeto> <id>, que é sub-tipo de T_ORDEM e onde <id> é um identificador-para diferenciar as diferentes chaves ou ordens de um mesmo objeto.

As instâncias definidas para os objetos do diagrama E-R formam uma classe, identificada por <T_ELEMENTO>. Cada entidade ou relacionamento especificada no esquema E-R gera um objeto T_<objeto>, que é sub-tipo de T_ELEMENTO.

Finalmente, o banco de dados como um todo também é considerado como uma classe, identificada por T_BANCO.

Assim, observamos que uma classe, neste contexto, é considerada um tipo de objeto e para cada classe existe um conjunto de métodos vistos pelos usuários do servidor E-R, que possibilitam a inicialização, acesso e término de manipulação de cada objeto. É importante ressaltar que existe também um conjunto de métodos que não aparecem para o usuário e servem para manipulação interna de arquivos da base de dados.

VI.2.3 – MÉTODOS DE ACESSO

As rotinas geradas visam proporcionar um suporte ao nível conceitual, possibilitando o desenvolvimento de aplicações que utilizam o banco sem o conhecimento de detalhes internos do mesmo. Isto é, foi feita uma separação entre os níveis físico e conceitual : o nível físico é controlado pelas rotinas de manipulação e o nível conceitual é controlado pelos programas de aplicação construídos ou pelas transações criadas pelo Gerador de Aplicações. Para o controle do nível físico, utiliza-se o TURBO PASCAL DATABASE TOOLBOX [PASC 19871, que necessita ser configurado para cada banco de dados gerado.

Para isolar os programas de aplicação ou as transações geradas de detalhes internos do banco, a manipulação do mesmo deve obedecer os seguintes critérios :

. arquivos de dados ou de índices não devem ser manuseados e sim os objetos do esquema E-R;

. todos os controles operacionais decorrentes da utilização do banco devem ser tratados automaticamente pelas rotinas de manipulação.

Para que isso aconteça, apenas uma parte dos métodos e objetos gerados fica visível para ser utilizado nos programas e transações. Esse subconjunto será detalhado a seguir.

- O objeto T_BANCO é usado para operar o banco como um todo. Possui os seguintes métodos:

  . **OPEN** : preparação para acesso;

  . **CLOSE** : término de acesso;

  . **CREATE** : geração da base de dados, com instâncias vazias.

- O objeto T_<objeto> é usado para realizar as operações nos arquivos de dados. Possui os seguintes métodos:

  . **STORE :** adiciona uma instância do objeto no arquivo de dados correspondente;

  . **UPDATE** : atualiza uma instância do objeto no arquivo de dados correspondente;

  . **DELETE :** remove uma instância do objeto do arquivo de dados correspondente;

  . **INSTANCIA** : retorna uma instância do objeto a partir do surrogate;

  . **IS_<sub-entidade> :** se <objeto> for super-entidade, este método é gerado para cada sub_entidade sua. Este método retorna o valor lógico verdadeiro se a instância de T <objeto> também for instância de T_<sub-entidade>; caso contrário retorna o valor falso;

  . <ordem> : este método é gerado para cada ordem ou chave com origem em <objeto>. Retorna uma variável que permite o acesso direto a ordem do objeto.

- O objeto T_ORDEM é usado para realizar os acessos a base de dados, segundo critérios de ordenação ou de chaves que são definidos no esquema E-R. Possui os seguintes métodos:

  . **RESET** : posiciona no início da ordem;

  . **RESETBACK** : posiciona no fim da ordem;

. **NEXT** : obtém o próximo elemento da ordem;

. **EOO** : testa o fim da ordem;

. **FIND** : utilizado na busca de um elemento que contenha atributo igual a determinada chave;

. **SEARCH** : utilizado na busca de um elemento que contenha atributo igual ou maior a determinada chave;

VI.3 – **INTERFACE** COM O USUÁRIO

A **interface** com o usuário para se efetivar as transações de alteração, exclusão e inclusão foi implementada utilizando-se o TURBO VISION [VISI 1990].

O TURBO VISION proporciona um suporte para aplicações que precisam de uma **interface** com o usuário interativa, flexível e de alta performance. O TURBO VISION oferece uma infra-estrutura básica, composta de uma biblioteca orientada a objetos, que pode ser usada para qualquer aplicação. Esta biblioteca contém:

a) suporte para **mouse**;

b) janelas múltiplas com tamanhos variáveis;

c) janelas especiais para diálogo, chamadas de "caixas de diálogo";

d) instalação de cores;

e) menus;

f) manipulação padrão de teclas e do **mouse**.

A comunicação com o usuário é feita através de uma ou mais janelas ou "caixas de diálogo", que aparecem e desaparecem na tela em resposta a comandos originários do **mouse** ou do teclado. As janelas podem sofrer rolamento e as "caixas de diálogo" podem conter "botões", que são palavras em destaque que podem ser selecionadas originando a transmissão de comandos para a aplicação.

Com o TURBO VISION, a realização das transações é feita de uma maneira bastante amigável para o usuário, que visualiza telas racionais e uniformes e executa as operações que desejar de maneira simples, direta e não ambígua.

É importante ressaltar que no decorrer de uma propagação de transação, a qualquer momento antes de sua concretização, a

transação pode ser cancelada através do comando ESC que retorna o controle ao manu principal.

Nos exemplos de utilização encontrados no apêndice, será possível vislumbrar algumas das potencialidades proporcionadas pelo TURBO VISION.

VI.4 – ESTRUTURA DO GERADOR DE TRANSAÇÕES

Como já foi visto anteriormente, o Gerador de Transações aqui descrito é parte componente do QUICK-DB. O QUICK-DB está implementado em TURBO PASCAL 6.0, com aproximadamente 35.000 linhas de código. A parte de manipulação gráfica foi especialmente desenvolvida, de modo a oferecer desempenho aceitável mesmo em equipamentos de tipo **xt.**

O módulo Gerador de Transações foi implementado em TURBO PASCAL 6.0, sendo constituído de dezessete sub-módulos: um deles correspondendo ao programa principal e cada um dos outros dezesseis correspondendo a uma das seguintes **units:**

. **PRINCIPAL (PRINCIPA.PAS)** – é o programa principal do módulo gerador de transações. É responsável pela ordem de geração e pela invocação de todas as **units.**

. **INTERFACE_IDENTIFICACAO (ANTEIDEN.PAS)** – é responsável pela geração da **interface** da **unit** que realiza a identificação de instâncias.

. **SELECIONA** (**SELECAO.PAS**) – gera os procedimentos responsáveis pela confirmação da instância de um relacionamento.

. **IDENTIFICACAO_INSTANCIA** (IDENTIFI.PAS) – gera os procedimentos responsáveis pela identificação da instância de uma entidade ou relacionamento.

. **IDENTIFICACAO_CHAVE (IDENTREL.PAS)** – gera os procedimentos responsáveis pela escolha da chave de acesso de uma entidade ou relacionamento.

. **INTERF'ACE_PROGRAMA (ANTEIMPL.PAS)** – é responsável pela geração da **interface** do programa principal do Gerador de Transações.

. **INCLUSA0 (INCLUSAO.PAS)** – gera os procedimentos de inclusão de uma instância de entidade ou relacionamento.

. **INCLUSAO_CONTINUACAO (PRECISA.PAS)** – gera a tela que informa que a inclusão da instância necessita ser propagada e pede confirmação para a propagação.

. **PROPAGACAO (PROPAGA.PAS**) – gera os procedimentos responsáveis pela propagação da inclusão.

INCLUSA0_RELACIONAMENTO (INCLREL.PAS) – gera os procedimentos de definição das instâncias das entidades envolvidas na inclusão de um relacionamento.

. **INICIO_EXCLUSAO (INICEXC.PAS)** – gera os procedimentos iniciais na transação de exclusão.

EXCLUSAO_RELACIONAMENTO (EXCLREL.PAS) – gera os procedimentos responsáveis pela propagação da exclusão da instância de um relacionamento.

. **EXCLUSAO (EXCLUSAO.PAS)** – é responsável pela geração dos procedimentos de exclusão de instâncias.

. **ALTERACAO (ALTERACA.PAS)** – é responsável pela geração dos procedimentos de alteração de instâncias.

. **INCLUSA0_ENTIDADE_RELACIONAMENTO (INCENTRL.PAS)** – é responsável pela geração do procedimento que vai controlar todas as transações de inclusão e suas possíveis propagações.

. **MANIPULACAO (HANDLE.PAS)** – gera os procedimentos de manipulação da ação do usuário através da **interface** TURBO VISION.

. **MENU_PRINCIPAL (PROGRAMA.PAS)** – gera o menu principal do programa Gerador de Transações.

Dentro do diagramador E-R, há uma opção para que o usuário possa gerar as transações referentes ao esquema especificado. O fonte do programa é composto do programa principal denominado <banco>_AP e de uma **unit** denominada <banco>_U1, onde <banco> é o nome do diagrama.

Depois, fora do diagramador E-R, para compilar as rotinas de manipulação e gerar o banco de dados vazio, se executa o comando GERACAO <banco>. Para compilar o programa contendo as transações de atualização, se executa o comando COMPILA <banco>.

Após estes passos, para que se possa utilizar as transações geradas, basta executar o comando <banco>_AP.

VI.5 – CONSISTÊNCIAS E ERROS DAS TRANSAÇÕES

Na realização das transações, algumas consistências são feitas automaticamente e em caso de erro aparece uma mensagem explicativa e o usuário tem a oportunidade de modificar os dados errados para que a transação possa ser executada com sucesso.

Os erros podem ser classificados em duas categorias: a) erros que acontecem dependendo de um determinado estado do banco de dados e b) erros que acontecem independente do estado atual do banco de dados.

Os erros que ocorrem independente de estados do banco são detectados pelas consistências locais e são os seguintes:

. Na inclusão dos atributos de uma instância, na alteração dos atributos de uma instância ou no preenchimento dos atributos de uma chave para identificação de uma instância, os atributos de tipo inteiro e real só podem ser preenchidos com valores numéricos; caso contrário, aparecerá a mensagem de erro: <atributo> **INVÁLIDO** e o usuário terá a oportunidade de corrigir o erro. <atributo> é o nome do atributo.

. No preenchimento dos atributos de uma chave para identificação de uma instância, todos os atributos da chave devem ser preenchidos; caso contrário, aparecerá a mensagem de erro : <atributo> **INDEFINIDO** para cada um dos atributos não preenchidos e o usuário terá a oportunidade de preencher os campos indefinidos.

. Na inclusão dos atributos de uma instância de entidade, todos os atributos que sejam componentes de alguma chave desta entidade ou sejam componentes de alguma chave própria de relacionamento devem ser preenchidos; caso contrário, aparecerá a mensagem de erro: <atributo> **INDEFINIDO** para cada um destes atributos não preenchidos e o usuário terá a oportunidade de preencher os campos indefinidos. Esta consistência é realizada devido à restrição denominada INTEGRIDADE DE ENTIDADE que já foi comentada no capítulo anterior.

. Na inclusão dos atributos de uma instância de relacionamento, todos os atributos que sejam componentes de alguma chave própria deste relacionamento devem ser preenchidos; caso contrário, aparecerá a mensagem de erro: <atributo> **INDEFINIDO** para cada um destes atributos não preenchidos e o usuário terá a oportunidade de preencher os campos indefinidos. Esta consistência é realizada devido à restrição denominada INTEGRIDADE DE ENTIDADE que já foi comentada no capítulo anterior.

Os erros que acontecem dependendo de um determinado estado do banco de dados são detectados pelas consistências globais e são os seguintes:

. Na identificação de uma instância de entidade ou relacionamento, após o preenchimento dos atributos da chave, se não existe nenhuma instância que corresponda àquela determinada chave, aparece a mensagem de erro: <**entidade**> **INEXISTENTE** e o usuário tem a oportunidade de preencher novos atributos componentes da chave de acesso. <entidade> é o nome da entidade ou relacionamento.

. Na inclusão ou alteração de uma instância, se já existe uma outra instância com o mesmo conteúdo de alguma chave, aparece a mensagem de erro: CHAVE <nomechave> DUPLICADA e o usuário pode corrigir os atributos e tentar realizar a inclusão/alteração novamente. <nomechave> é o nome da chave que ficaria repetida se a transação fosse realizada. Esta consistência é realizada devido à propriedade das chaves denominada UNICIDADE DE CHAVE que já foi comentada no capítulo anterior.

. Na inclusão de uma instância do relacionamento R, ao se definir a instância da entidade E1 participante do relacionamento R, esta instância pode já estar relacionada a alguma instância da entidade E2 através de R. Nesse caso, se Cmax (vide RESTRIÇÕES DE INTEGRIDADE RELATIVAS no capítulo anterior) for igual a 1, aparecerá a mensagem de erro: <entidade1> NÃO PODE TER MAIS DE UMA <entidade2> RELACIONADA e o usuário terá a oportunidade de definir uma outra instância da entidade E1 para participar do relacionamento. <entidade1> é o nome da entidade E1 e <entidade2> é o nome da entidade E2.

VI.6 – ABRANGÊNCIA

Devido a limitação de tempo, foi definido um escopo de abrangência para este trabalho que restringe a utilização de determinadas construções na especificação do esquema E-R, a saber:

. **Toda entidade deve possuir ao menos uma chave (esta chave pode ser herdada).** Esta restrição acontece porque a implementação dos procedimentos de identificação da instância de uma entidade foi feita através das chaves desta entidade. Esta restrição não se aplica aos relacionamentos, porque se estes não tiverem chave própria, pode-se utilizar as chaves das entidades envolvidas para identificação da instância.

. Toda entidade deve possuir ao menos um atributo (este atributo pode ser herdado). Restrição conseqüente da restrição acima, visto que uma chave é composta de pelo menos um atributo.

. Todo relacionamento deve possuir ao menos um atributo. Esta restrição acontece por dois motivos: a) na confirmação da identificação da instância de um relacionamento (ao se usar chaves das entidades envolvidas), são mostrados os atributos de cada instância do relacionamento que corresponde as chaves fornecidas e b) ao se mostrar as instâncias do relacionamento que vão ser excluídas, são mostrados os atributos desta instância. Para se evitar esta restrição, a solução nos dois casos seria mostrar os atributos de alguma chave de ambas as entidades envolvidas.

. Atributos do tipo lógico ainda não foram implementados no Gerador de Transações. A solução seria a implementação de um campo de tamanho 1 em que o usuário teria que preencher V ou F na inclusão ou na alteração.

. Atributos definidos como mandatory (obrigatório) no esquema E-R ainda não foram implementados. A implementação seria análoga aos atributos que são componentes de chaves: devem ser obrigatoriamente preenchidos.

. Entidades associativas (agregados) e entidades fracas ainda não foram implementadas. Por isso existe a restrição de especificação destas estruturas no esquema E-R.

. Na atual versão do Gerador de Transações, as inclusões de entidades devem se iniciar nas entidades do nível mais baixo em uma generalização/especialização. Esta restrição faz com a especificação de relacionamentos com entidades envolvidas que não estão no nível mais baixo em uma generalização/especialização possa causar problemas para o Gerador de Transações.

. O Gerador de Transações não trata a possibilidade de atributos multivalorados, ou seja, considera-se que o banco gerado a partir do mapeamento entre o modelo E-R e o modelo relaciona1 básico (já mencionado anteriormente neste trabalho) já esta normalizado.

. O Gerador de Transações não trata relacionamentos que não sejam binários. Esta restrição é pouco importante, pois relacionamentos de grau maior do que dois não são muito utilizados. As restrições de cardinalidade nos relacionamentos de grau maior do que dois são muitas vezes ambíguas e neste caso estes relacionamentos devem ser substituídos por entidades associativas.

. Na atual versão do Gerador de Transações, as cardinalidades no esquema podem assumir os valores 0 ou 1 para Cmin (cardinalidade mínima) e os valores 1 ou N para Cmax (cardinalidade máxima). Para eliminar esta restrição, deveria ser possível que Cmin e Cmax assumissem valores constantes.

CAPÍTULO VII - CONCLUSÕES

O objetivo deste capítulo é realizar uma análise conclusiva dos principais pontos abordados nesta dissertação. São apresentadas as contribuições obtidas, são sugeridas algumas possíveis extensões ao presente trabalho e finalmente, são feitos comentários comparativos a respeito de outros trabalhos correlatos.

VII.1 - CONTRIBUIÇÕES DO PROJETO

Uma das premissas adotadas por ocasião da definição do projeto UniversiData foi a de que as soluções pudessem ser integralmente implementadas, com resultados práticos imediatos. Há, inclusive, uma experiência piloto no NCE/UFRJ de utilização do QUICK-DB na implementação do Sistema de Biblioteca, onde a equipe de desenvolvimento da aplicação está trabalhando em conjunto com a equipe do projeto do QUICK-DB.

Quanto ao estágio atual de desenvolvimento do UniversiData, vários dos módulos estão concluídos e outros em andamento. As partes do servidor, dicionário de dados, interface para sistemas especialistas, interface para programas de aplicação e gerador de aplicações estão concluídas.

Conclui-se, então, que o Gerador de Transações aqui proposto faz parte de um ambiente parcialmente implantado e que está operacional. Este ambiente é bastante propício para pesquisas e desenvolvimento, pois já possui um conjunto razoável de ferramentas que facilitam a experimentação de técnicas adicionais.

O mérito do QUICK-DB é reunir sob um mesmo paradigma de utilização facilidades como: diagramação E-R, interfaces com sistemas especialistas, folheadores, geração de aplicação e dicionários de dados. Com isso, é oferecido ao usuário um ambiente completo, uniforme e consistente para o desenvolvimento de aplicações de bancos de dados.

VII.2 - CONTRIBUIÇÕES DO GERADOR DE TRANSAÇÕES

Em seu atual estágio, o Gerador de Transações está operacional, sendo capaz de gerar automaticamente programas para os tipos de transação mais usuais como inclusão, exclusão e alteração de entidades e relacionamentos. Nestas operações, são respeitadas as propriedades do esquema E-R tais como cardinalidades, propagação de inclusão e exclusão, chaves e atributos obrigatórios. O Gerador de Transações utiliza a

biblioteca de métodos do TURBO PASCAL 6.0 para a definição de interfaces, gerando programas compactos, elegantes, robustos e com interfaces extremamente elaboradas.

As classes de objetos que formam o banco de dados são acessadas por métodos contidos no servidor E-R. Estes métodos, também chamados de rotinas de manipulação, isolam os programas de aplicação e as transações geradas de detalhes internos do banco.

Verifica-se que o Gerador de Transações utiliza-se destas rotinas para a criação das transações de atualização, obtendo outras rotinas com um nível de abstração mais elevado. As transações foram implementadas aproveitando o suporte para a interface com o usuário oferecido pelo TURBO VISION. Os módulos componentes do programa Gerador de Transações são descritos e são mostradas as consistências que são verificadas durante a realização das transações.

Concluindo, com a atual versão do Gerador de Transações, é possível gerar aplicações simples para manutenção dos bancos de dados, através de um conjunto de transações voltado para o Modelo Entidade-Relacionamento.

VII.3 – CONTINUIDADE DO TRABALHO

A importância do Gerador de Transações reside não somente nos resultados descritos anteriormente, mas também em seu potencial de continuidade. Há várias extensões num trabalho desta natureza, entre as quais podemos citar as seguintes.

Uma primeira categoria de extensões seria aumentar o escopo de abrangência mencionado no capítulo anterior. Assim, as extensões desta categoria permitiriam que se eliminasse a restrição de utilização de determinadas construções na especificação do esquema E-R. Por exemplo, seria possível a especificação de entidades associativas, visto que o Gerador de Transações já teria o padrão destas embutido em seu código para poder implementar transações envolvendo entidades associativas.

VII.3.1 – GERADOR DE RELATÓRIOS

Esta extensão permitiria que, através da mesma interface TURBO VISION, o usuário pudesse especificar quais dados seriam obtidos dos arquivos e como seria a sua formatação nos relatórios de saída. O diálogo com o usuário se daria através de cardápios em que o usuário teria apenas que responder SIM ou NÃO ou escolher entre várias opções já pré-selecionadas.

Devido ao fato dos geradores de relatório serem linguagens não procedurais, o usuário teria então de especificar o que ele gostaria que ele fosse feito, ao passo que o sistema (gerador de relatório) resolveria como realizar a tarefa.

VII.3.2 – MÓDULO DE CONSULTAS

O projeto UniversiData já possui o seu módulo de consultas, que permite que se faça a consulta através do próprio esquema E-R. A idéia desta extensão seria oferecer uma outra alternativa de consultar a base de dados, desta vez aproveitando a interface TURBO VISION que já é utilizada pelo Gerador de Transações. Dessa forma, tanto a expressão da consulta por parte do usuário, quanto a exibição dos dados consultados, aproveitariam a infra-estrutura de cardápios, janelas, mouse, cores e "caixas de diálogo" provida pelo TURBO VISION.

VII.3.3 – INTERFACE WINDOWS

Segundo alguns especialistas, o sistema operacional WINDOWS [WIND 1990] está substituindo o DOS a passos rápidos. Já existe uma versão do TURBO PASCAL, o NEW TURBO PASCAL FOR WINDOWS, que é um **software** que possui o padrão WINDOWS e permite que o programador construa o seu programa PASCAL já com este padrão.

Uma possível extensão deste trabalho seria a implementação de uma nova versão do Gerador de Transações utilizando-se o TURBO PASCAL FOR WINDOWS (ao invés do TURBO PASCAL 6.0) e que gerasse transações com uma interface WINDOWS com o usuário (ao invés da interface TURBO VISION), de maneira que as transações geradas pudessem ter o mesmo padrão que está se popularizando pelo mundo.

VII.3.4 – REUTILIZAÇÃO DAS TRANSAÇÕES POR PROGRAMAS

As rotinas de manipulação podem ser utilizadas diretamente por programas de aplicação, de maneira que estes passam a ser independentes em relação ao SGBD utilizado. Da mesma forma, as transações geradas pelo Gerador de Transações poderiam ser adaptadas para que também pudessem ser reutilizadas por programas de aplicação, que dessa forma não precisariam ter mecanismos para assegurar a integridade do banco, visto que a verificação de integridade seria automaticamente tratada pelas transações.

VII.3.5 - CONSISTÊNCIA DE ATRIBUTOS

Na atual versão do diagramador E-R, ao se especificar o esquema E-R, é possível definir o tipo de um atributo como sendo real, inteiro, alfanumérico ou lógico e esta definição de tipos é considerada na geração das transações.

Se fosse implementado no diagramador E-R uma definição de domínio para os atributos, isto é, os valores possíveis que o atributo poderia assumir; esta definição faria com que o preenchimento dos campos dos atributos nas transações de inclusão e alteração passasse por uma consistência de domínio.

VII.3.6 - MECANISMO DE **ESCAPE**

Uma outra extensão seria a implementação de novas informações variáveis no Gerador de Transações (além das contidas no esquema E-R) através do mecanismo de **escape.** O Gerador de Transações é capaz de resolver uma classe muito limitada de problemas (atualizações simples do banco de dados) e a possibilidade de implementar determinados procedimentos diretamente em TURBO PASCAL 6.0 aumentaria bastante o seu potencial de utilização. Assim, estes procedimentos arbitrários seriam colocados em trechos especificados no programa contendo as transações e seria possível que os procedimentos chamassem as transações ou fossem chamados por estas.

VII.3.7 - ESPECIFICAÇÃO FUNCIONAL

O projeto UniversiData permite uma especificação dos dados da aplicação, que consiste do esquema conceitual E-R. Uma extensão para o projeto UniversiData seria a possibilidade de especificar aspectos puramente funcionais do sistema, através de algum dos modelos funcionais existentes.

Com o aproveitamento conjunto da especificação dos dados e da especificação funcional, seria possível que o Gerador de Transações passasse a ter uma utilização menos restrita, não se limitando apenas a gerar a aplicação para manutenção da base de dados, visto que determinadas funções da aplicação já seriam geradas junto com as transações.

VII.4 – ANÁLISE COMPARATIVA

Já foi visto no capítulo 2 que embora existam vários exemplos de geradores de aplicação em diferentes áreas, há pouca semelhança entre eles na forma como é realizada a tradução da especificação de entrada para o produto final.

Com base nos exemplos de outros geradores de aplicação encontrados na literatura, foi feita uma análise comparativa do Gerador de Transações, tendo se concluído que este apresenta vantagens e desvantagens, que serão detalhadas a seguir:

VANTAGENS

. O fato de estar inserida numa ferramenta maior, o QUICK-DB, que possui vários componentes interligados, caracterizando um ambiente de desenvolvimento de **software** integrado. O Gerador de Transações, por exemplo, recebe como entrada o componente Dicionário de Dados e utiliza as rotinas de manipulação geradas pelo componente Gerador de Rotinas de Manipulação para poder acessar o banco de dados.

. A utilização das rotinas de manipulação pelo Gerador de Transações citada acima acarreta a vantagem de simplificação da implementação do Gerador de Transações; considerando que não houve a preocupação com detalhes internos do banco de dados, que já são tratados pelas rotinas de manipulação.

. Como conseqüência da vantagem descrita acima, temos também a independência em relação ao SGBD utilizado. Se o SGBD escolhido fosse outro, o Gerador de Transações não sofreria alteração alguma, as rotinas de manipulação é que deveriam ser modificadas.

. Nos geradores de aplicações encontrados na literatura, se constrói uma especificação que somente vai servir como entrada para o gerador de aplicações; no nosso caso, se define uma especificação que é utilizada principalmente na modelagem conceitual dos dados da aplicação e que também serve como entrada para o Gerador de Transações.

. O Gerador de Transações recebe como entrada o esquema E-R especificado. Isso faz com que se tenha necessariamente uma especificação dos dados do sistema junto com a aplicação gerada. E esta especificação vai estar sempre atualizada, já que para se realizar uma modificação na aplicação, é preciso primeiro alterar o esquema que lhe dá origem.

. A especificação de entrada para o Gerador de Transações (o esquema E-R) apresenta algumas vantagens em relação a outros tipos de especificação devido às seguintes características da

modelagem E-R: a) o modelo é simples, de maneira que o esquema é facilmente construído e é facilmente compreensível para os usuários, b) o modelo é bastante difundido e amplamente aceito e c) os conceitos do esquema apresentam uma interpretação única devido aos conceitos formais do modelo.

. O programa gerado permite que se realize as transações através de uma interface com o usuário bastante amigável, simples, direta e não ambígua; com a utilização de recursos como janelas, **mouse**, cardápios, "caixas de diálogo", manipulação padrão de teclas e outros.

. O banco de dados vai estar sempre íntegro após a realização de uma transação. Isso acontece devido ao grande enfoque que se coloca na verificação das restrições de integridade. Dessa forma, o usuário não precisa se preocupar com a possibilidade de sua transação deixar o banco inconsistente, pois a aplicação foi implementada com o objetivo de evitar que isto aconteça.

DESVANTAGEM

. A desvantagem básica do Gerador de Transações é que seu alcance de utilização é bastante restrito, considerando que ele só é capaz de realizar a manutenção de bases de dados. Foi por este motivo que este módulo recebeu esta denominação, ao invés de ser chamado de Gerador de Aplicações. Para abrandar esta desvantagem, foram feitas as sugestões de extensão da seção anterior: a) mecanismos de escape, b) especificação funcional como entrada para o gerador de aplicações e c) utilização das transações geradas por programas de aplicação.

Assim, acredita-se que a principal contribuição desta tese foi justamente a proposta de um protótipo de gerador de aplicações simples, prático e aberto a extensões; integrando conhecimentos de modelagem de dados, restrições de integridade e geração de aplicações. Tudo isso inserido num ambiente completo de projeto e manipulação de bancos de dados construídos com base no modelo Entidade-Relacionamento.

REFERÊNCIAS BIBLIOGRÁFICAS

[ANSI 1975] Tsichritzis, D.C. e Klug, A. (eds.) "The ANSI/X3/SPARC DBMS framework report of the study group on database management systems". Information Systems, Oxford, Pergamon Press, 3, pp. 173-191, 1978.

[BALZ 1985] Balzer, R., "A 15 Year Perspective on Automatic Programming". IEEE Transactions on Software Engineering, Vol. 11, no. 11, pp. 1257-1268, November 1985.

[BARN 1991] Barnes, B.H. e Bollinger, T.B., "Making Reuse Cost-Effective". IEEE Software, pp. 13-24, January 1991.

[BIGG 1984] Biggerstaff, T. e Perlis, A.J. (eds.), "Foreword of the Special Issue on Software Reusability". IEEE Transactions on Software Engineering, Vol. 10, no. 5, pp. 474-476, September 1984.

[BIGG 1987] Biggerstaff, T. e Richter, C., "Reusability Framework, Assessment, and Directions". IEEE Software, pp. 41-49, March 1987.

[BURN 1986] Luker, P.A. e Burns, A., "Program Generators and Generation Software". The Computer Journal, Vol. 29, no. 4, pp. 315-321, 1986.

[BURN 1988] Allen, P. e Burns, A., "Program Generation for Ada - A Case Study". Software - Practice and Experience, Vol. 18, no. 12, pp. 1125-1138, December 1988.

[CERI 1992] Batini, C. ; Ceri, S. e Navathe, S. B., "Conceptual Database Design : An Entity-Relationship Approach". The Benjamin/Cummings Publishing Company, Inc., 1992.

[CHEA 1984] Cheatham, T.E., "Reusability Through Program Transformations". IEEE Transactions on Software Engineering, Vol. 10, no. 5, pp. 589-594, September 1984.

[CHEN 1976] Chen P.P., "The Entity-Relationship Model : Toward a Unified View of Data". ACM Transactions on Database Systems, vol. 1, no. 1, March 1976, pp. 9-36.

[CHEN 1977] Chen P.P., "The Entity-Relationship Model : A Basis for the Enterprise View Of Data". Proceedings IFIPS NCC 46, no. 46, 1977, pp. 76-84.

[CHNG 1984] Cheng, T.T., Lock, E.D., Prywes, N.S., "Use of Very High Level Languages and Program Generation by Management Professionals". IEEE Transactions on Software Engineering, Vol. 10, no.5, pp. 552-563, September 1984.

[CLEA 1988] Cleaveland, J.C., "Building Application Generators". IEEE Software, pp. 25-33, July 1988.

[CODD 1979] Codd E.F., "Extending the Database Relational Model to Capture More Meaning". ACM Transactions on Database Systems, vol. 4, no. 4, December 1979, pp. 397-434.

[CODD 1981] Codd E.F., "Data Models in Database Management". ACM SIGPLAN Notices, vol. 16, no. 1, January 1981.

[DATE 1985] Date C.J., "An Introduction to Database Systems, Volume II". Addison-Wesley Publishing Company, Inc., 1985.

[DATE 19861 Date C.J., "An Introduction to Database Systems, Volume I". Addison-Wesley Publishing Company, Inc., Fourth Edition, 1986.

[DELV 1991] Delvaux, M.M., "Geração de Rotinas Orientadas a Objetos para Manipulação de um Banco de Dados". Projeto Final de Curso, Instituto de Matemática, UFRJ, 1991.

[DIAZ 1987] Díaz, R.P., "Classifying Software for Reusability". IEEE Software, pp. 6-16, January 1987.

[FARI 1991] Faria, E.C., "MARTE : Meta-gerador de Aplicações baseado na Reutilização de Templates". Tese de Mestrado, Programa de Engenharia de Sistemas e Computação, COPPE-UFRJ, Setembro 1991.

[HORO 1984] Horowitz, E. e Munson, J.B., "An Expansive View of Reusable Software". IEEE Transactions on Software Engineering, Vol. 10, no. 5, pp. 477-487, September 1984.

[HORO 1985] Horowitz, E., Kemper, A., e Narasimhan, B., "A Survey of Application Generators". IEEE Software, pp. 40-54, January 1985.

[JONE 1984] Jones, T.C., "Reusability in Programming: A Survey of the State of the Art". IEEE Transactions on Software Engineering, Vol. 10, no. 5, pp. 488-494, September 1984.

[KERN 1984] Kernighan, B.W., "The UNIX System and Software Reusability". IEEE Transactions on Software Engineering, Vol. 10, no.5, pp. 513-518, September 1984.

[KOGU 19911 Kogut, A. e Melo, P.C.M, "Um Gerador de Aplicações em Pascal Orientado a Objetos". Ferramentas de Software do V Simpósio Brasileiro de Engenharia de Software, pp. 9, Outubro 1991.

[KORT 1986] Korth, H.F. e Silberschatz A., "Sistemas de Bancos de Dados". McGraw-Hill, Inc., 1989.

[LANE 1984] Lanergan, R.G. e Grasso, C.A., "Software Engineering with Reuusable Design and Code". IEEE Transactions on Software Engineering, Vol. 10, no.5, pp. 498-501, September 1984.

[LENZ 1983] Lenzerini, M. e Santucci, G., "Cardinality Constraints in the E-R Model". Proceedings of the Third International Conference on Entity-Relationship Approach, pp. 529-549, October 1983.

[MASI 1991] Masiero, P.C. e Meira, C.A.A., "Um Gerador de Aplicações para Sistemas Reativos". Anais do IV Simpósio Brasileiro de Engenharia de Software, pp. 45-58, Outubro de 1991.

[MEYE 1987] Meyer, B., "Reusability: The Case for Object-Oriented Design". IEEE Software, pp. 50-64, March 1987,

[MEYE 19881 Meyer, B., "Object-Oriented Software Construction". Prentice-Hall International Ltd, 1988.

[NEIG 1984] Neighbors, J.M., "The Draco Approach to Constructing Software from Reusable Components". IEEE Transactions on Software Engineering, Vol. 10, no. 5, pp. 564-573, September 1984.

[PACI 1991] Pacitti, E.C., "Especificação e Implementação de um Sistema de Regras de Produção Fortemente Acoplado a Bancos de Dados", Tese de Mestrado, Programa de Engenharia de Sistemas e Computação, UFRJ, Outubro de 1991.

[PASC 1987] Borland International, Inc., "Turbo Pascal : Database Toolbox". Scotts Valley, Ca, 1987.

[PASC 1989] Borland International, Inc., "Turbo Pascal 5.5 : Object Oriented Programming Guide". Scotts Valley, Ca, 1989.

[PASC 1990] Borland International, Inc., "Turbo Pascal 6.0 : Programmer's Guide". Scottts Valley, Ca, 1990.

[PECK 1988] Peckham, J. e Maryanski, F., "Semantic Data Models". ACM Computing Surveys, New York, vol. 20, no. 3, pp. 153-189, September 1988.

[RICH 1988] Rich, C. e Waters, R.C., "Automatic Programming: Myths and Prospects". IEEE Computer, pp. 40-51, August 1988.

[RISH 1988] Rishe, N., "Database Design Fundamentals". Prentice-Hall International, Inc., 1988.

[SETZ 1989] Setzer, V.W., "Bancos de Dados". Editora Edgard Blucher LTDA, 1989.

[SILV 19881 Silveira, P.M., "Projeto de Banco de Dados Auxiliado por Computador". XXI Congresso Nacional de Informática, Rio de Janeiro, 1988.

[SILV 1989] Silveira, P.M., "A Formalization of the E-R Model". IX Conferencia Internacional de la Sociedad Chilena de Ciencia de la Computacion, Santiago, 1989.

[SILV 19901 Silveira, P.M., "Definindo e Utilizando Bancos de Dados com o Modelo Entidade-Relacionamento". XXIII Congresso Nacional de Informática, Rio de Janeiro, 1990.

[SILV 1992] Silveira, P.M., "QUICK-DB: Um Ambiente para Projeto e Manipulação de Bases de Dados". Relatório Técnico, Núcleo de Computação Eletrônica, UFRJ, 1992.

[SMIT 1977] Smith, J.M. e Smith, D.C.P., "Database Abstractions : Aggregation and Generalization". ACM Transactions on Database Systems, vol. 2, no. 2, June 1977, pp. 105-133.

[TAKA 1989] Takahashi, T., "O Paradigma de Objetos : Introdução e Tendências". IX Congresso da SBC, VIII Jornada de Atualização em Informática, Curso de Introdução a Programação Orientada a Objetos, UFU, 1989.

[TAKA 1990] Takahashi, T. e Liesenberg, H.K.E., "Programação Orientada a Objetos". VII Escola de Computação, São Paulo, 1990.

[TOSO 1990] Toso, V.E.F. e Silva, M.F., "Gerador de Banco de Dados do Quick-DB", Projeto Final de Curso, Instituto de Matemática, UFRJ, 1990.

[TRIN 1991] Trinkenreich, H., "Aspectos da Implementação de Interfaces Gráficas para Consultas a Bancos de Dados", Tese de Mestrado, Programa de Engenharia de Sistemas e Computação, COPPE, UFRJ, Maio de 1991.

[TSIC 19821 Tsichritzis, D.C. e Lochovsky, F.H., "Data Models". Prentice-Hall, Inc., 1982.

[ULLM 1980] Ullman, J.D., "Principles of Database Systems". Computer Science Press, Inc., 1980.

[UNIS 1984] Unisys Corporation, "DMS II - Data and Structure Definition Language (DASDL)", A Series, Dezembro de 1984.

[VISI 1990] Borland International, Inc., "Turbo Pascal 6.0 : Turbo Vision Guide". Scotts Valley, Ca, 1990.

[WIND 1990] Microsoft, "Windows User's Guide 3.0 : For The Windows Graphical Environment". Redmond , WA, 1990.

APÊNDICE - EXEMPLOS DE UTILIZAÇÃO

Para ilustrar este trabalho com alguns exemplos de utilização, vamos considerar o esquema exemplo retratado na figura 12.

Este esquema é referente à uma universidade, na qual, em cada período letivo, todo professor deve lecionar no máximo uma disciplina. As características de tal esquema serão descritas a seguir:

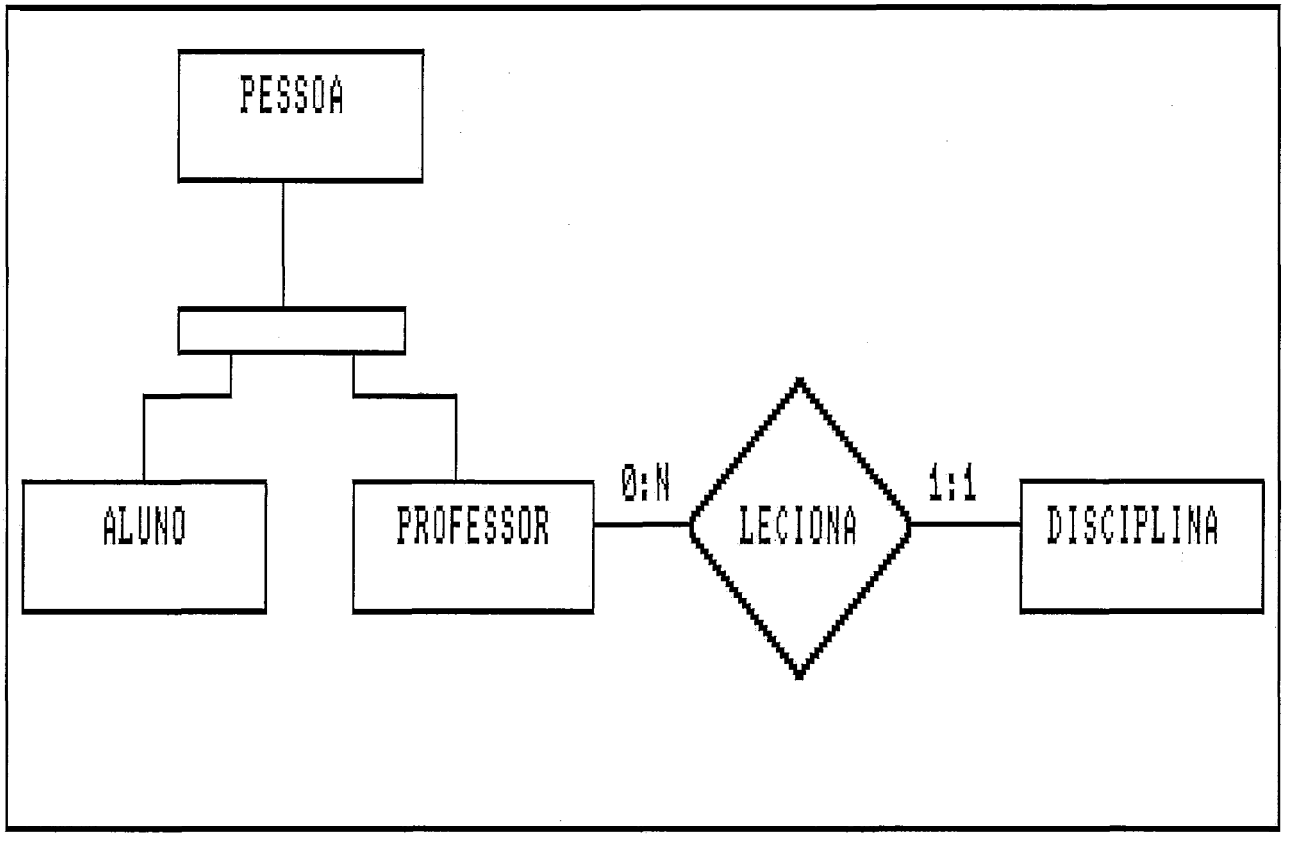


FIGURA 12 - ESQUEMA EXEMPLO

Entidade : PESSOA

Atributos : NOME, CPF, ENDEREÇO, TELEFONE

Chave : PORCPF (CPF)

Entidade : ALUNO

Atributos : DRE, CURSO

Chave : nenhuma

Entidade : PROFESSOR

Atributos : DEPARTAMENTO, REGISTRO

Chave : PORREGISTRO (REGISTRO)

Entidade : DISCIPLINA

Atributos : CODIGO, CREDITOS

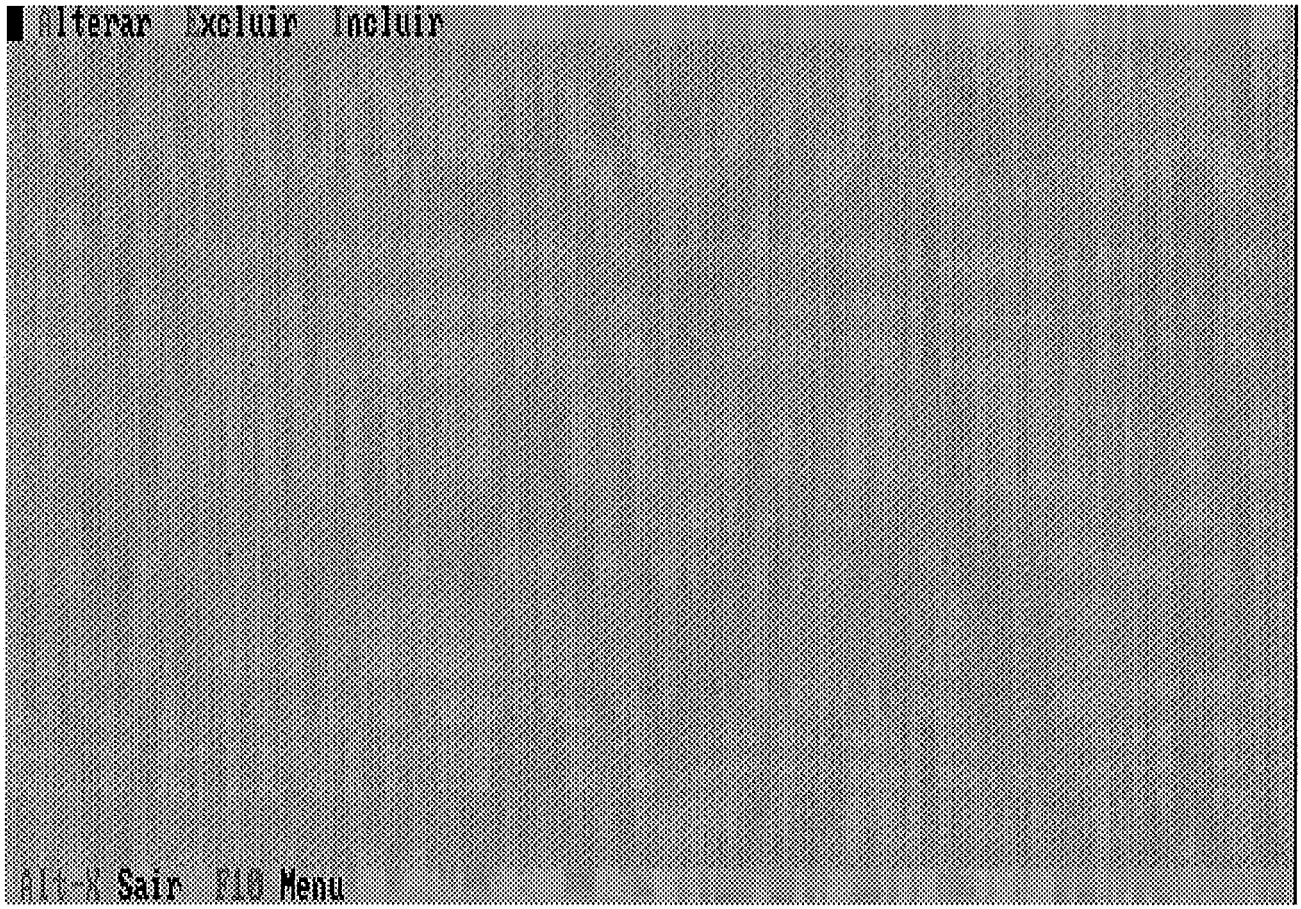
Chave : PORCODIGO (CÓDIGO)

Relacionamento : LECIONA

Atributos : TURMA, HORÁRIO

Chave : PORTURMA

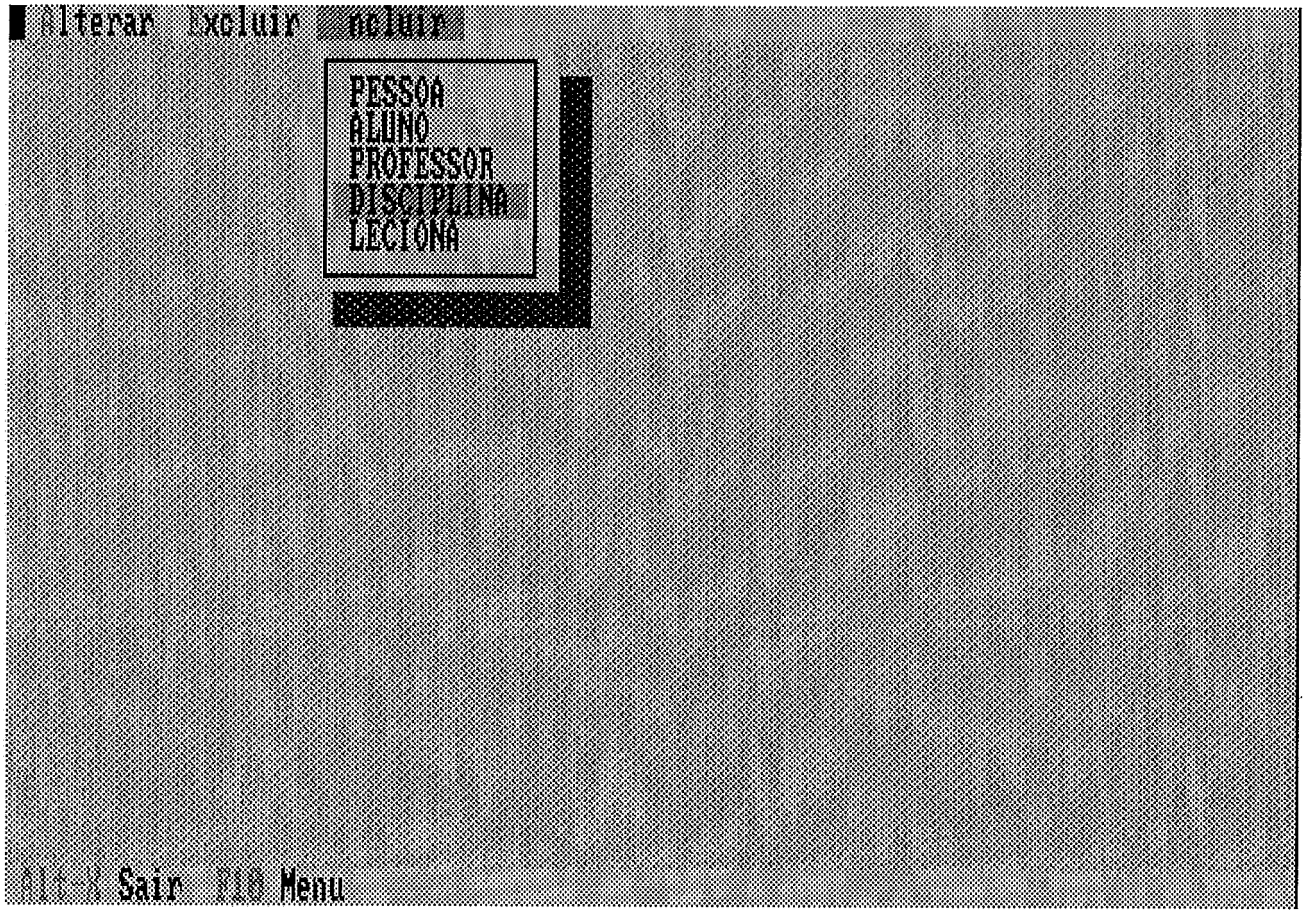
O cardápio principal da aplicação gerada referente as transações para este esquema é o seguinte:



TELA 0

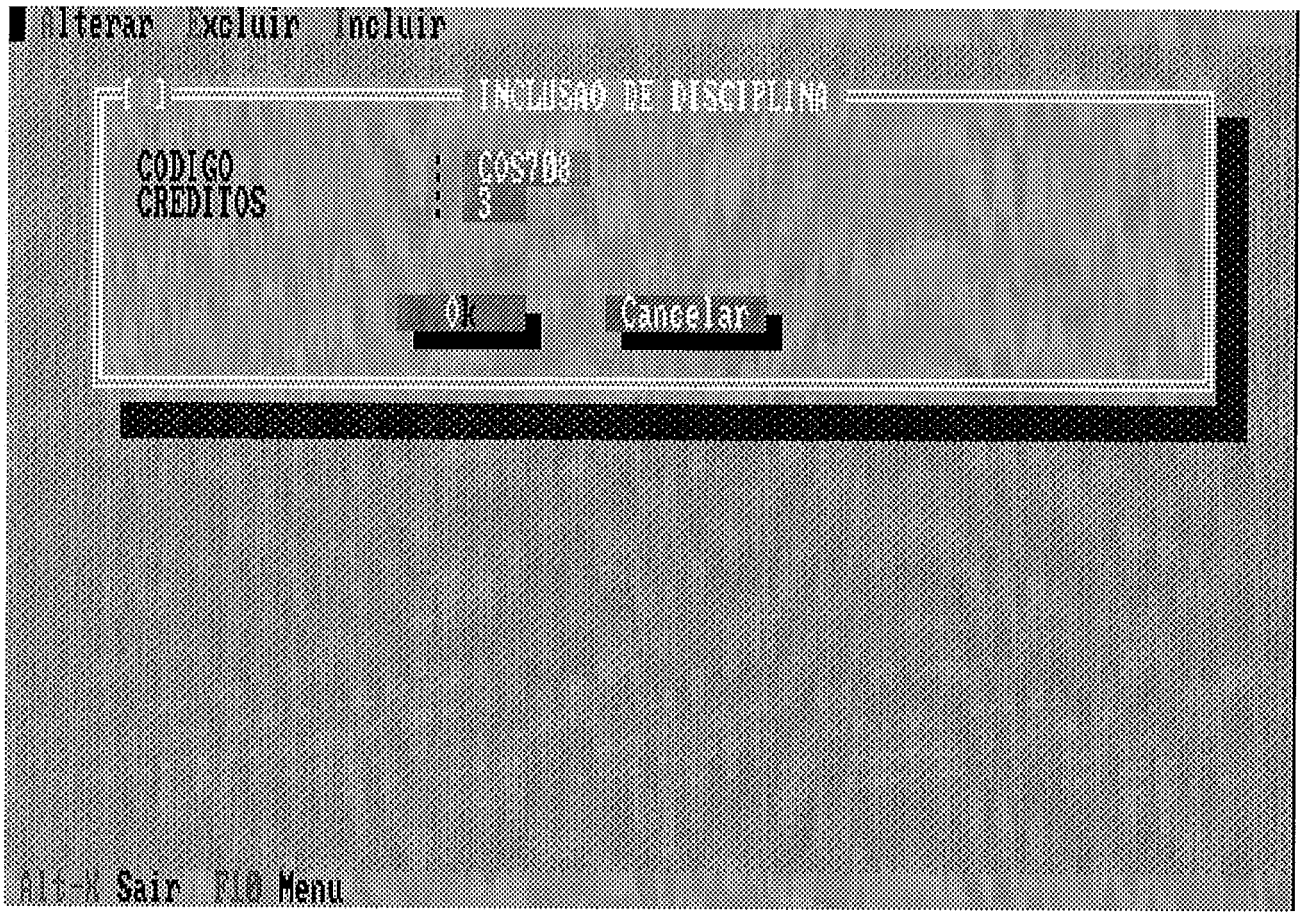
TRANSAÇÃO 1

O primeiro exemplo de transação é a inclusão de uma instância de DISCIPLINA. Após a escolha da opção INCLUSÃO, aparece o cardápio principal modificado:



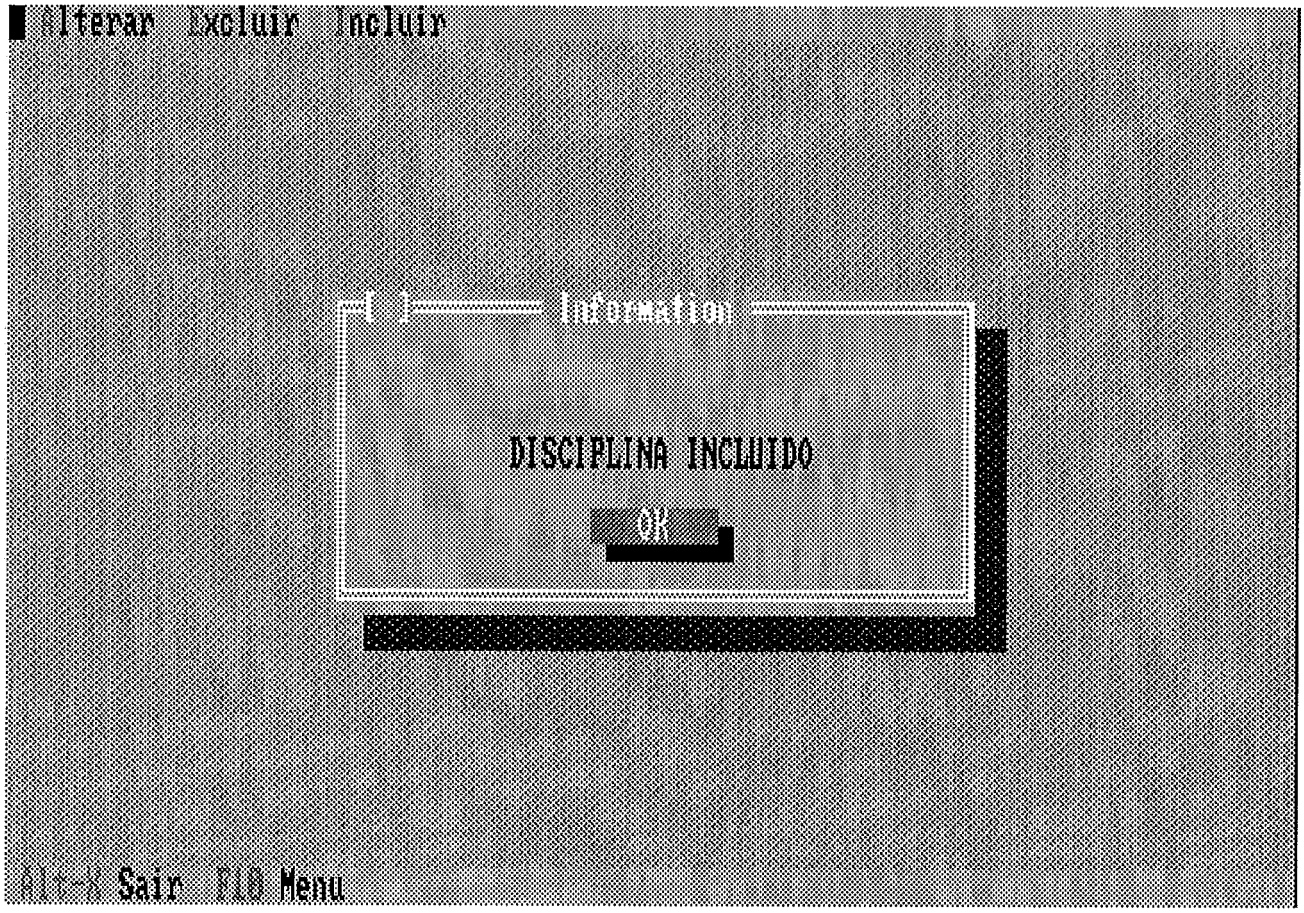
TELA 1.1

Após a escolha da opção DISCIPLINA, aparece a tela para se preencher os valores dos atributos de DISCIPLINA:



TELA 1.2

O usuário preenche os valores dos atributos de DISCIPLINA. Após a inclusão da instância de DISCIPLINA, aparece uma mensagem informando que a transação foi bem sucedida.



TELA 1.3

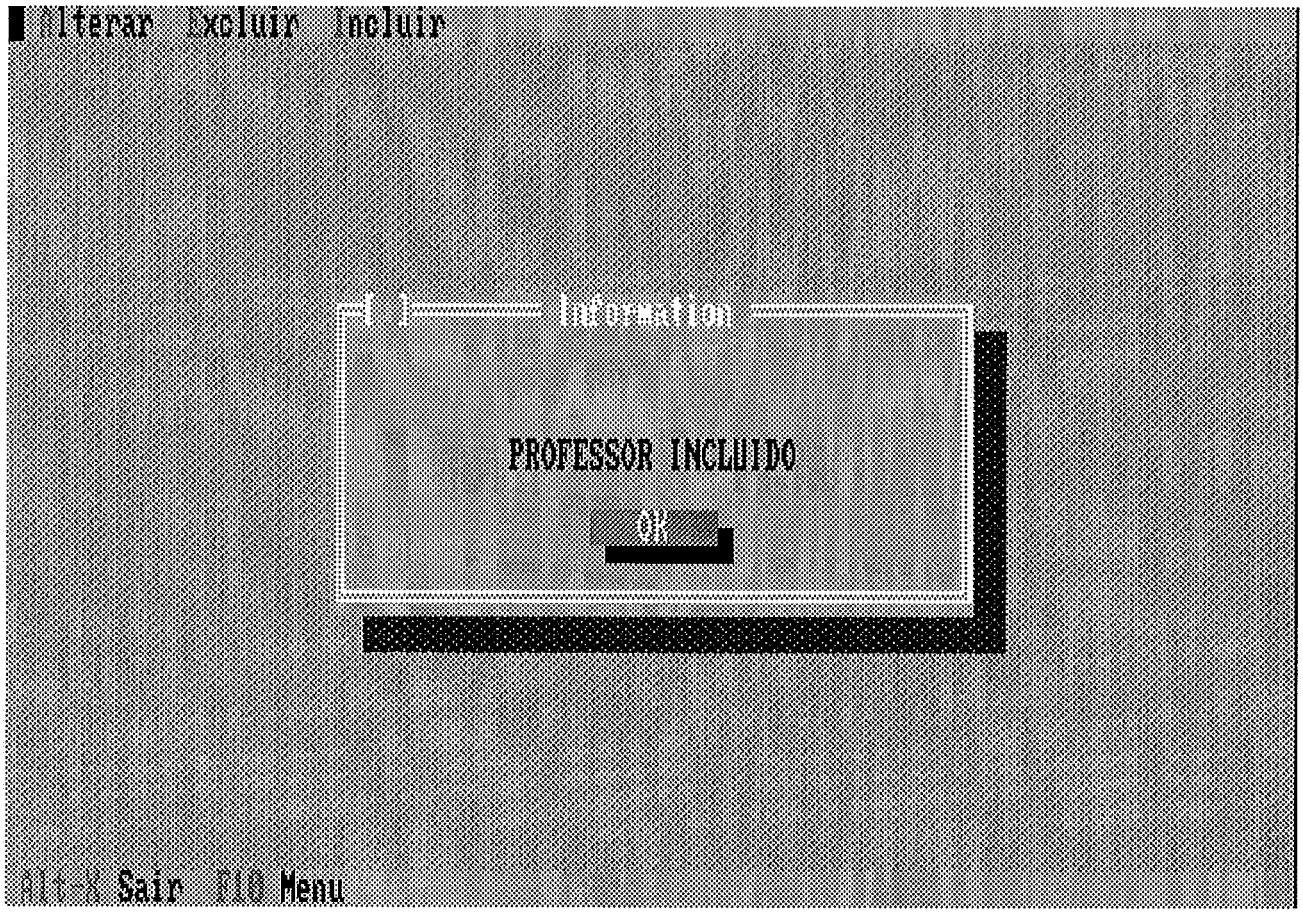
TRANSAÇÃO 2

O segundo exemplo de transação é a inclusão de uma instância de PROFESSOR. Após a escolha da opção INCLUSÃO, aparece a seguinte tela (VIDE TELA 1.1).

Após a escolha da opção PROFESSOR, aparece a tela para se preencher os valores dos atributos de PROFESSOR:

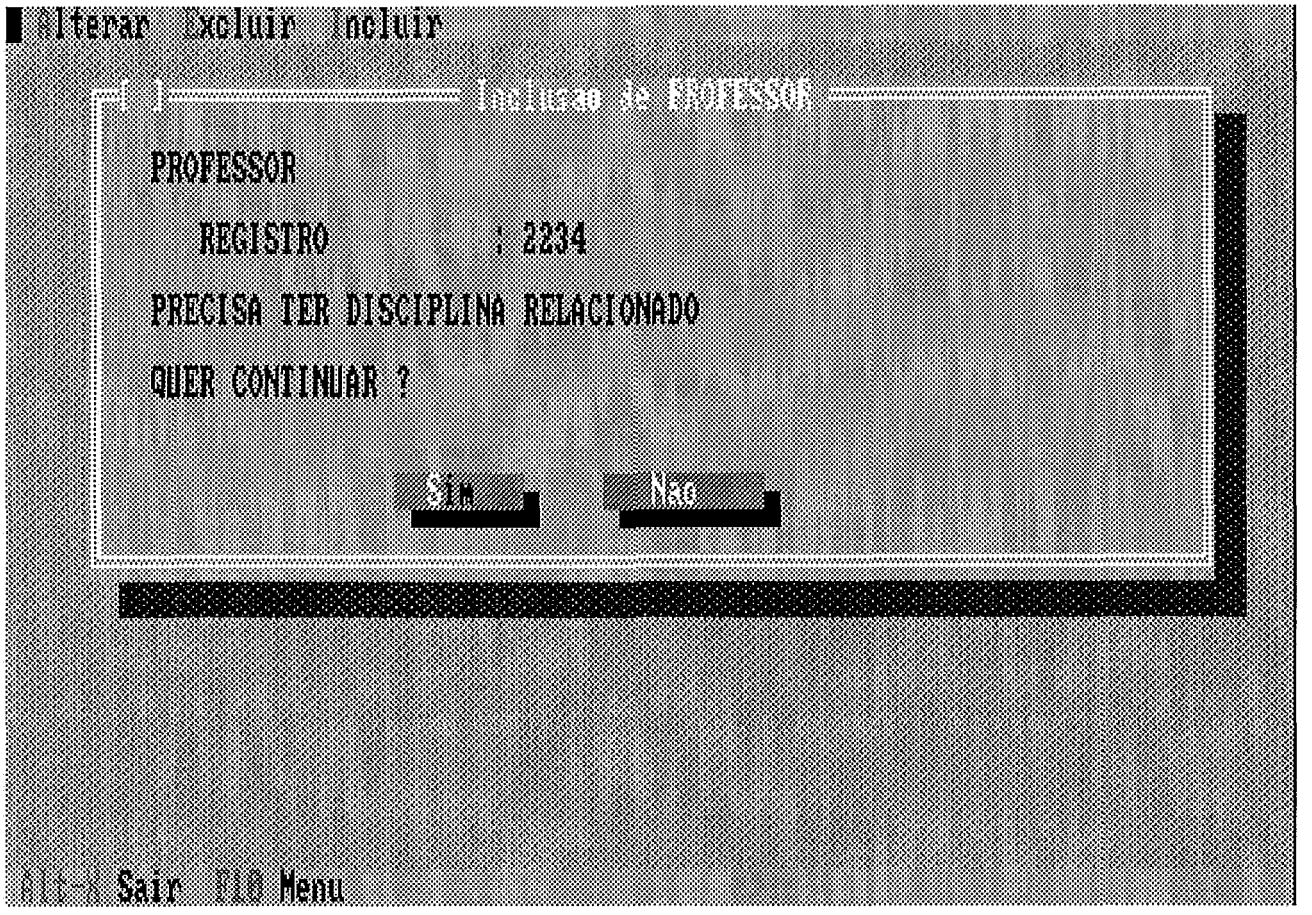
TELA 2.1

O usuário preenche os valores dos atributos de PROFESSOR. Após a inclusão da instância de PROFESSOR, aparece uma mensagem informando que a transação foi bem sucedida.



TELA 2.2

Depois, aparece uma tela informando que a inclusão da instância de PROFESSOR acarreta propagação e perguntando se a transação deve ser propagada ou cancelada:



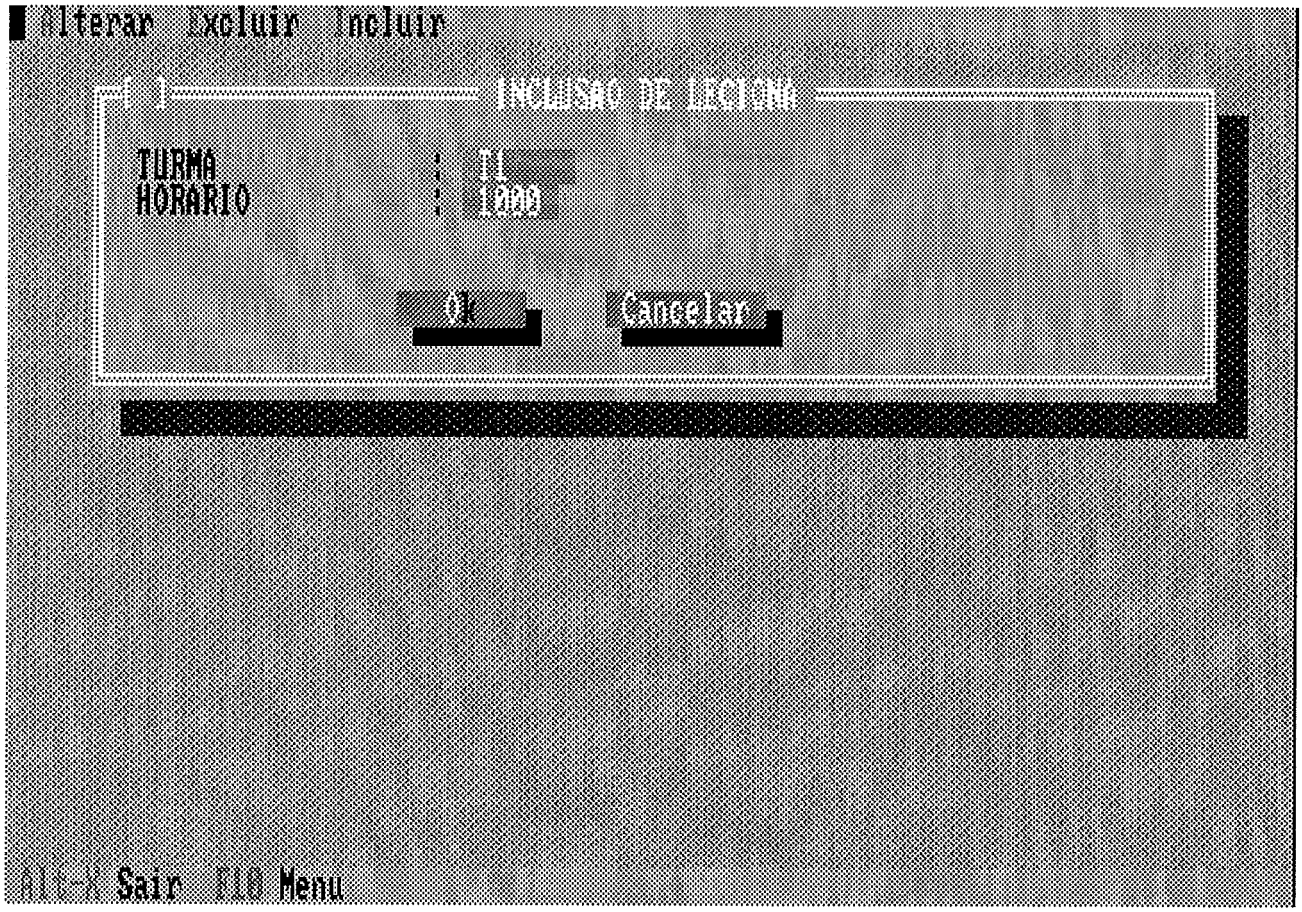
TELA 2.3

Após a escolha da opção de propagar a transação, aparece uma tela perguntando se a instância de DISCIPLINA que vai se relacionar com a instância de PROFESSOR já existe ou precisa ser incluída.

TELA 2.4

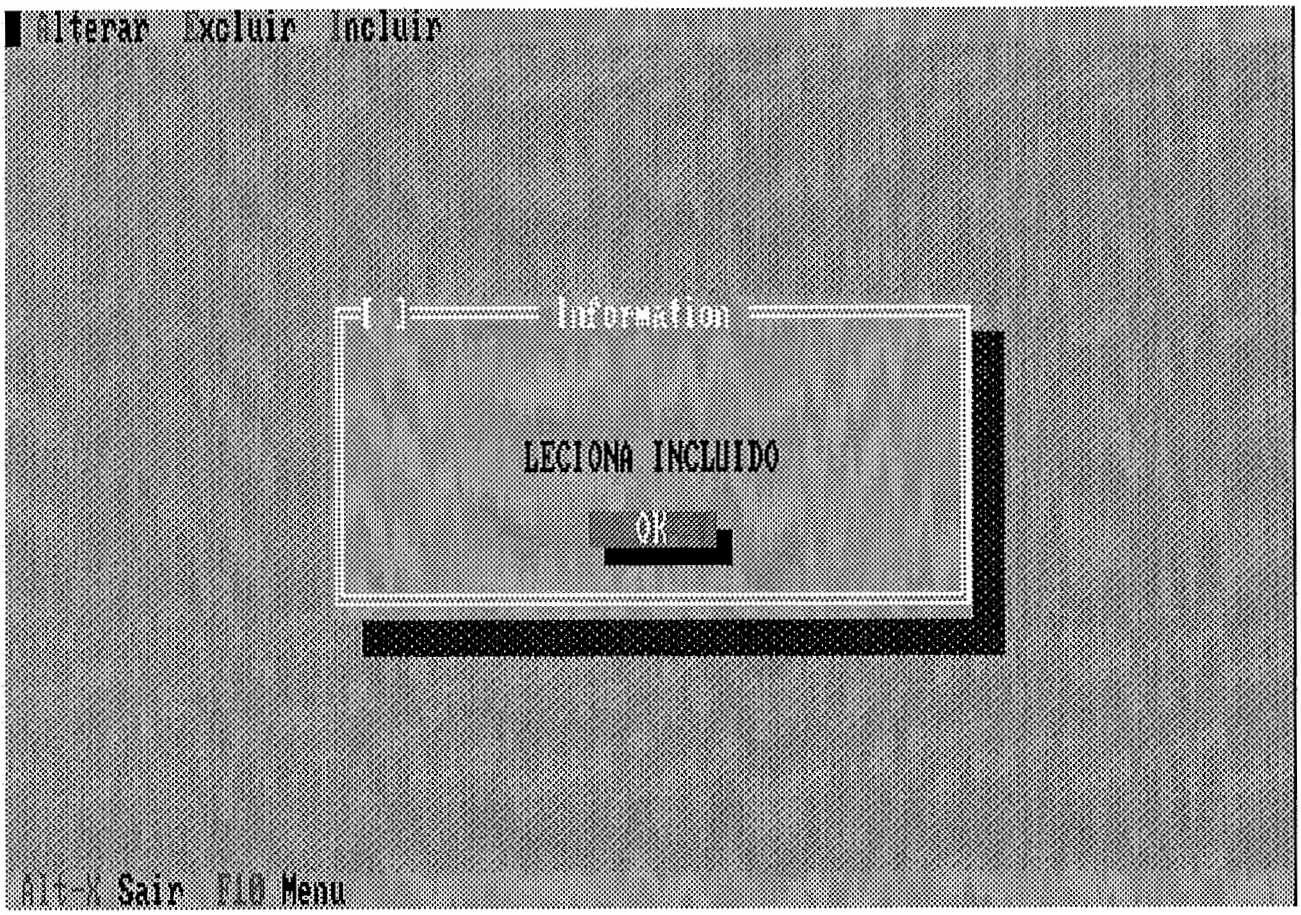
Após a escolha da opção de que a instância de DISCIPLINA precisa ser incluída, aparecem as mesmas telas 1.2 e 1.3 já vistas anteriormente.

Depois, aparece a tela para se preencher os valores dos atributos do relacionamento LECIONA:



TELA 2.5

O usuário preenche os valores dos atributos do relacionamento LECIONA. Após a inclusão da instância do relacionamento LECIONA, aparece uma mensagem informando que a transação foi bem sucedida.

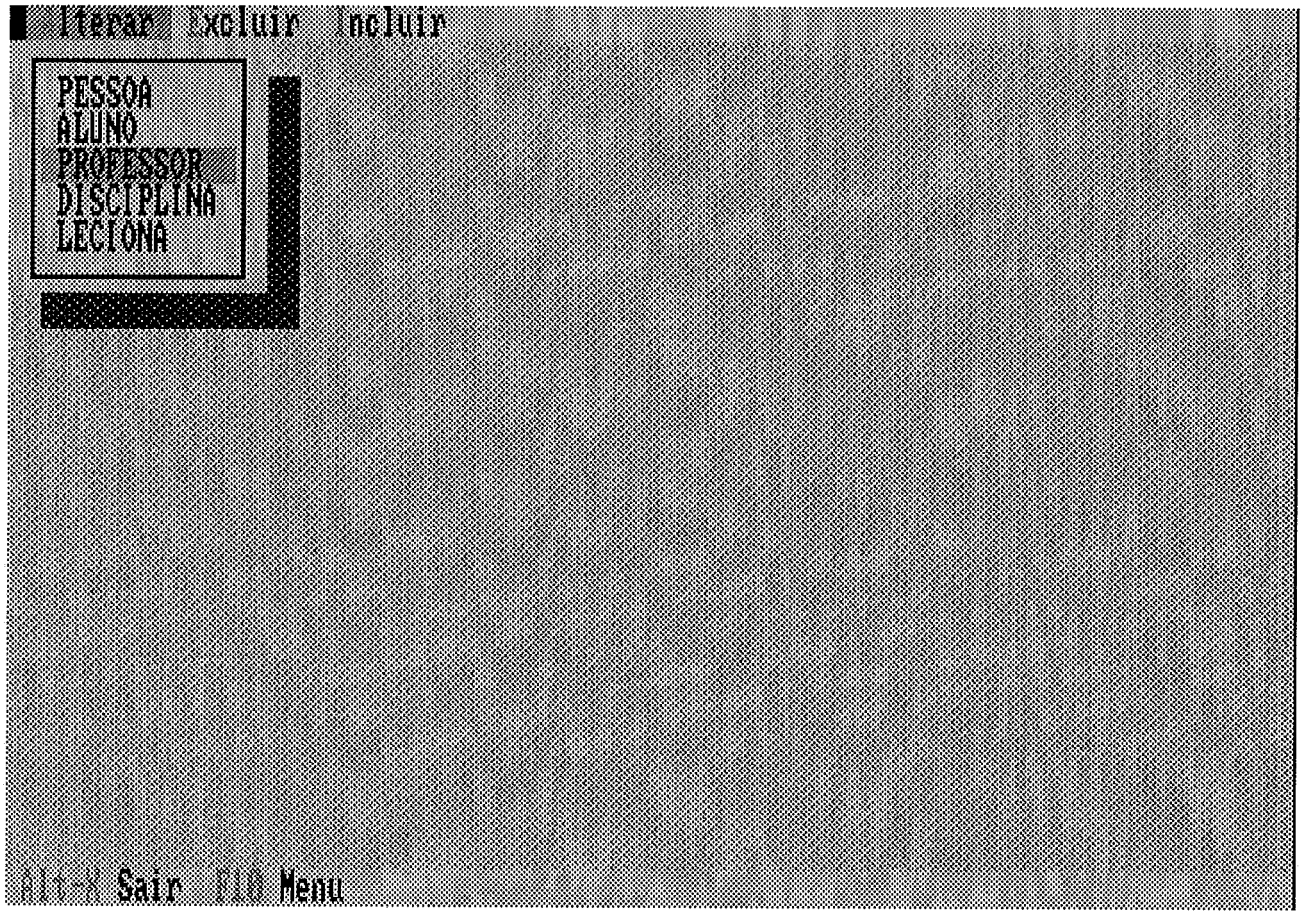


TELA 2.6

A macrotransação de inclusão foi bem sucedida. Esta macrotransação foi composta das microtransações de inclusão da instância de PROFESSOR, inclusão da instância de DISCIPLINA e inclusão da instância do relacionamento LECIONA.

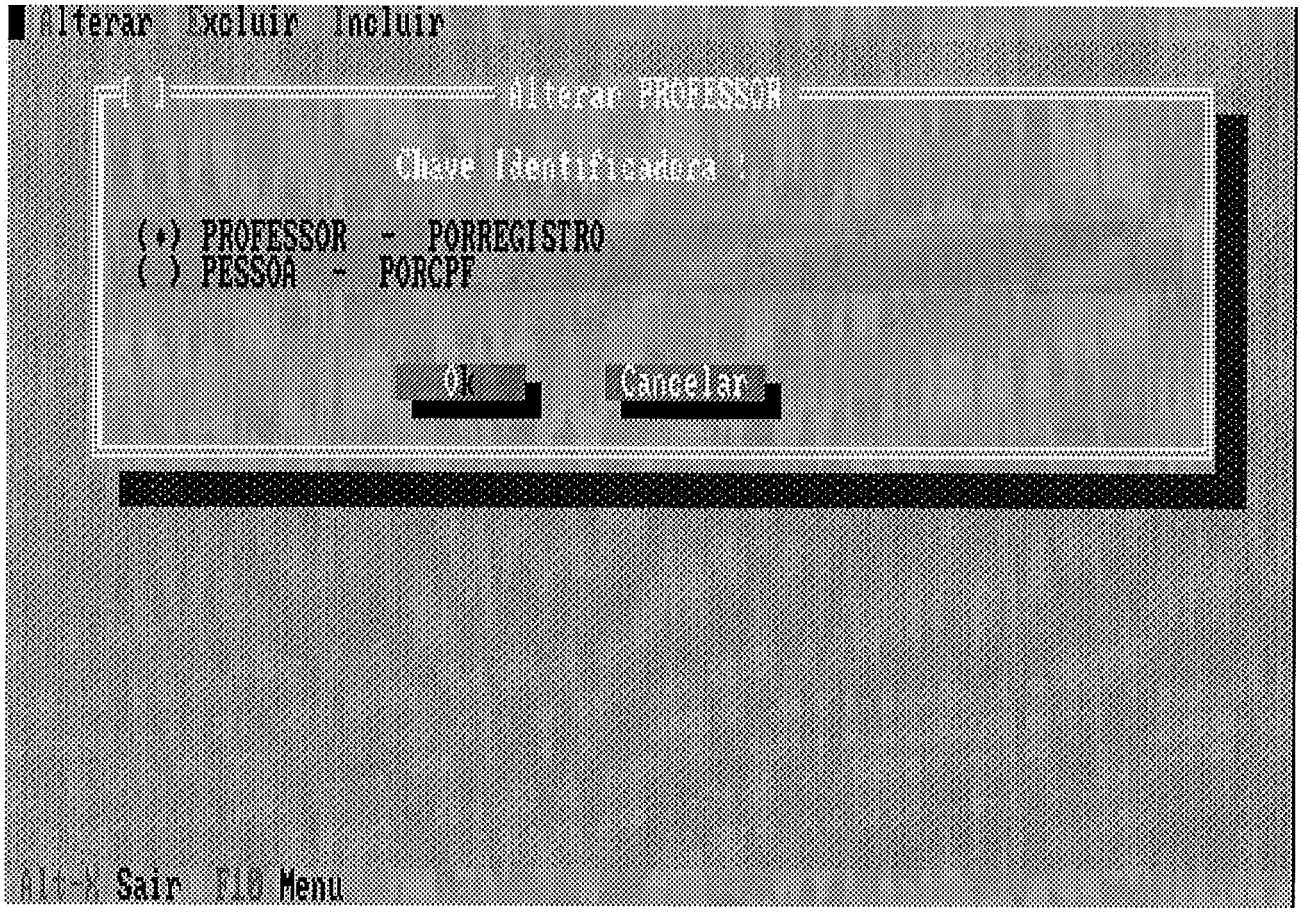
TRANSAÇÃO 3

O terceiro exemplo de transação é a alteração de uma instância de PROFESSOR. Após a escolha da opção ALTERAÇÃO, aparece a seguinte tela:



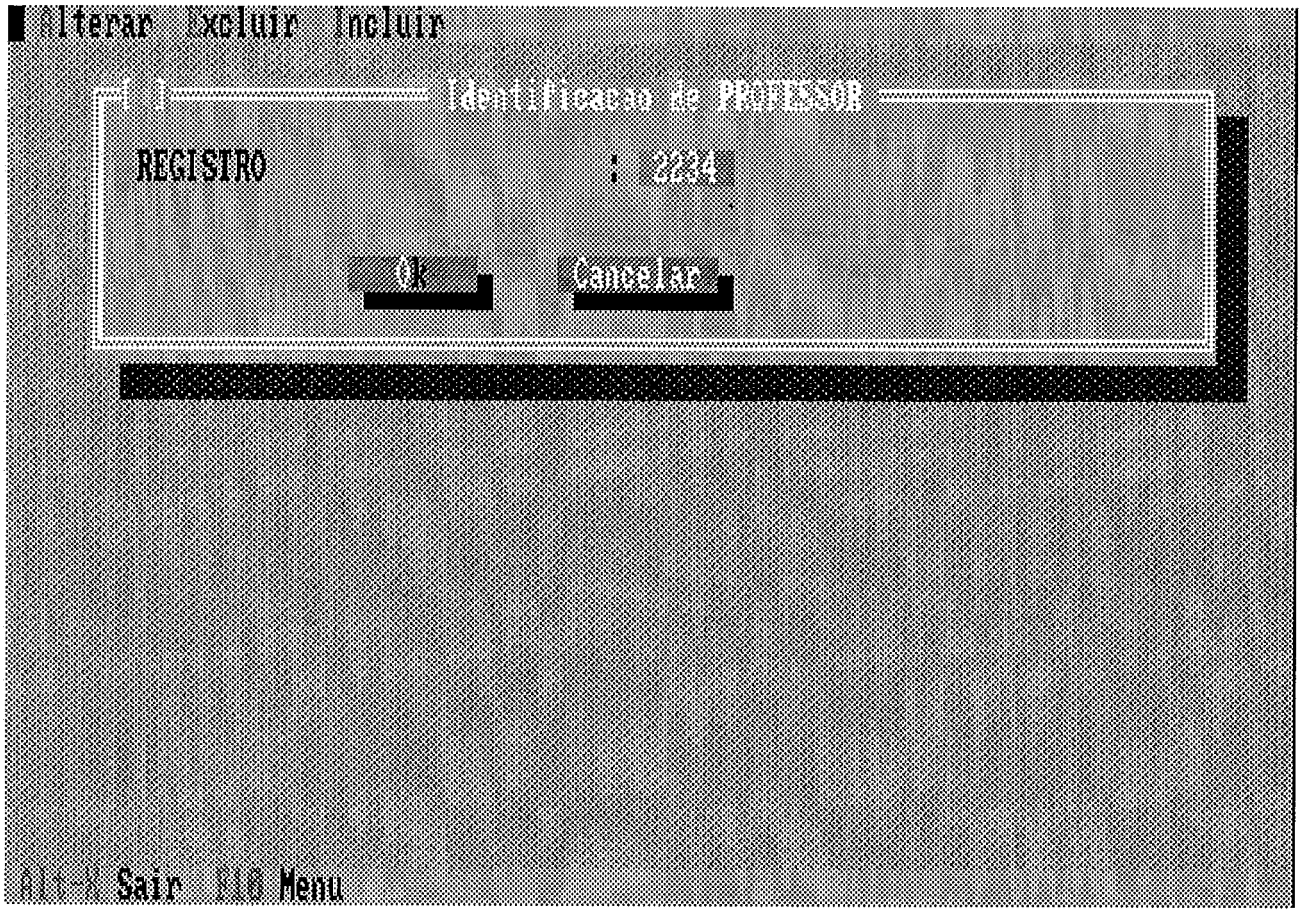
TELA 3.1

Após a escolha da opção PROFESSOR, aparece a tela onde se pode escolher a chave de acesso que vai ser utilizada para identificar a instância de PROFESSOR que vai ser alterada.



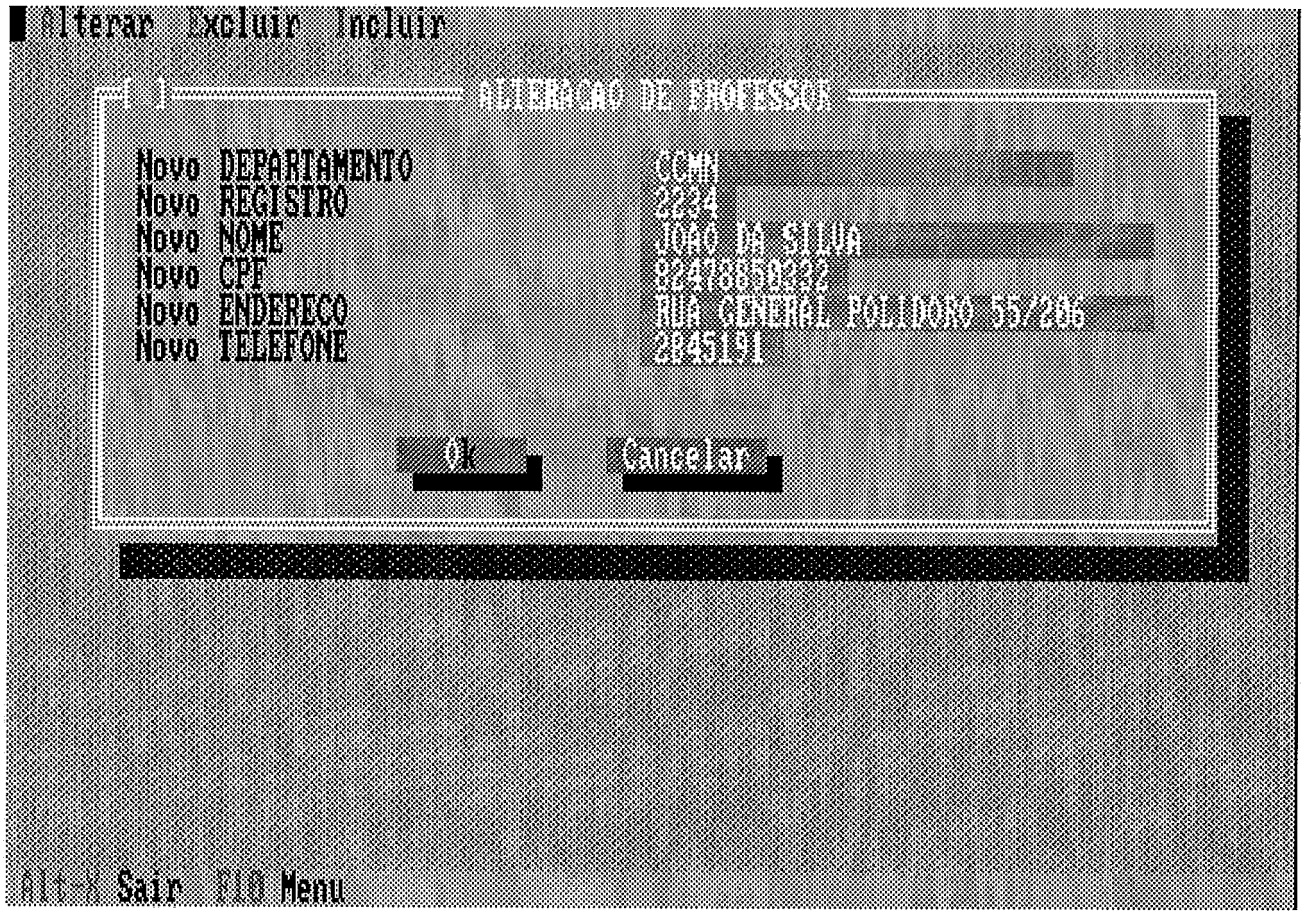
TELA 3.2

Após a escolha da chave de acesso PORREGISTRO, aparece a tela onde se preenche o valor do atributo REGISTRO:



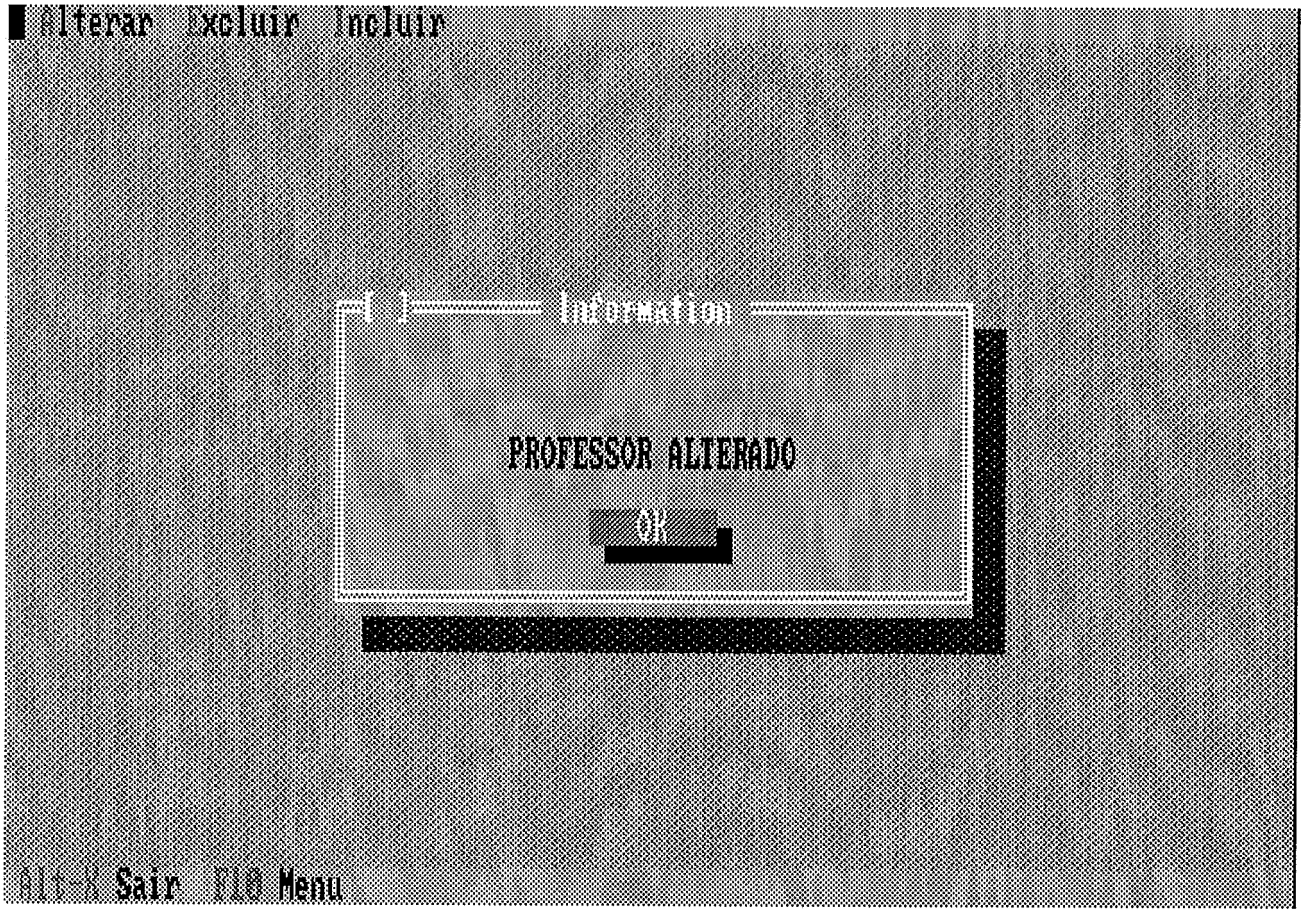
TELA 3.3

O usuário preenche o valor do atributo REGISTRO. Depois, aparece uma tela mostrando os valores atuais dos atributos da instância de PROFESSOR que foi acessada:



TELA 3.4

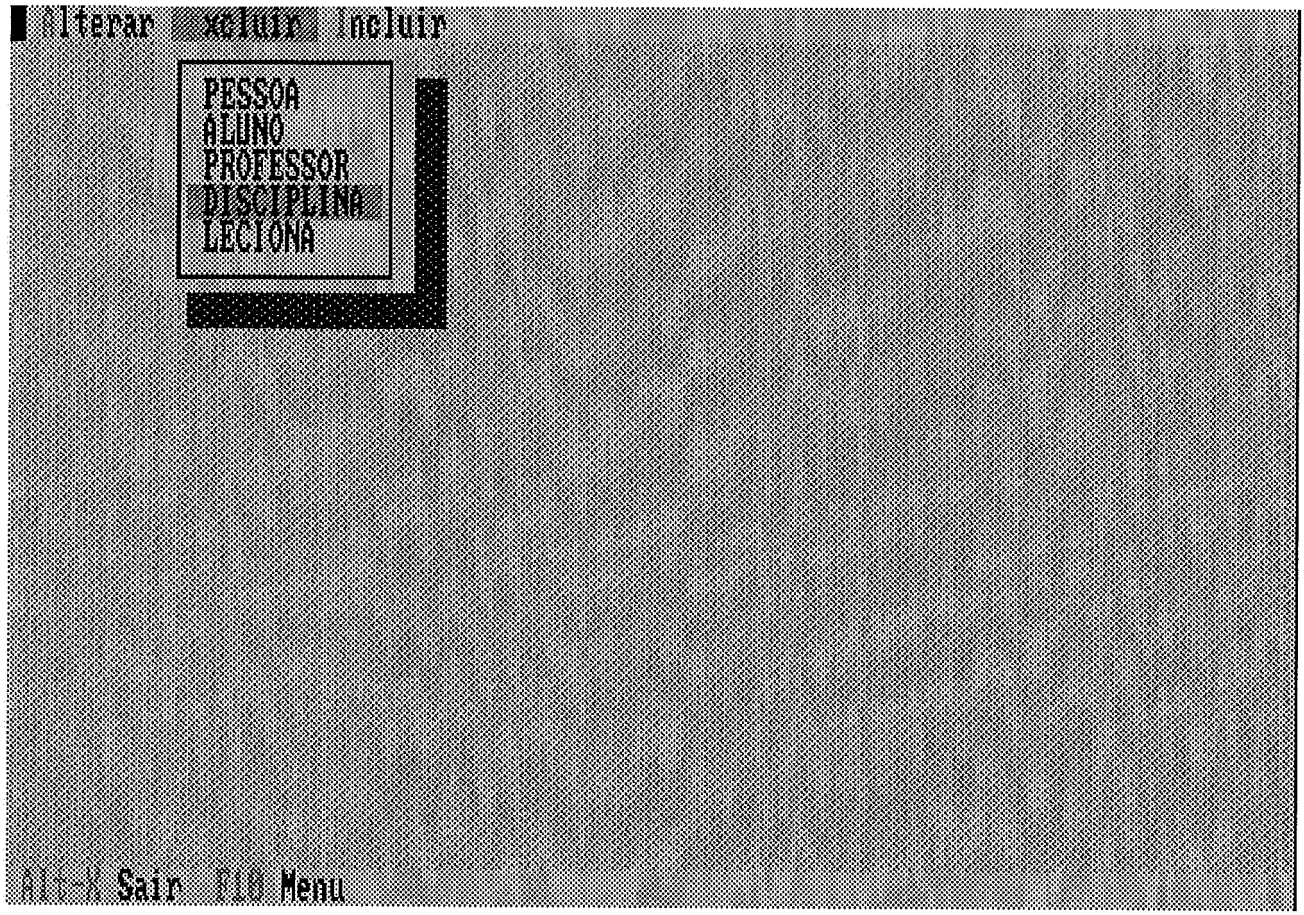
O usuário altera os campos que quiser. Após a alteração da instância de DISCIPLINA, aparece uma mensagem informando que a transação foi bem sucedida:



TELA *3.5*

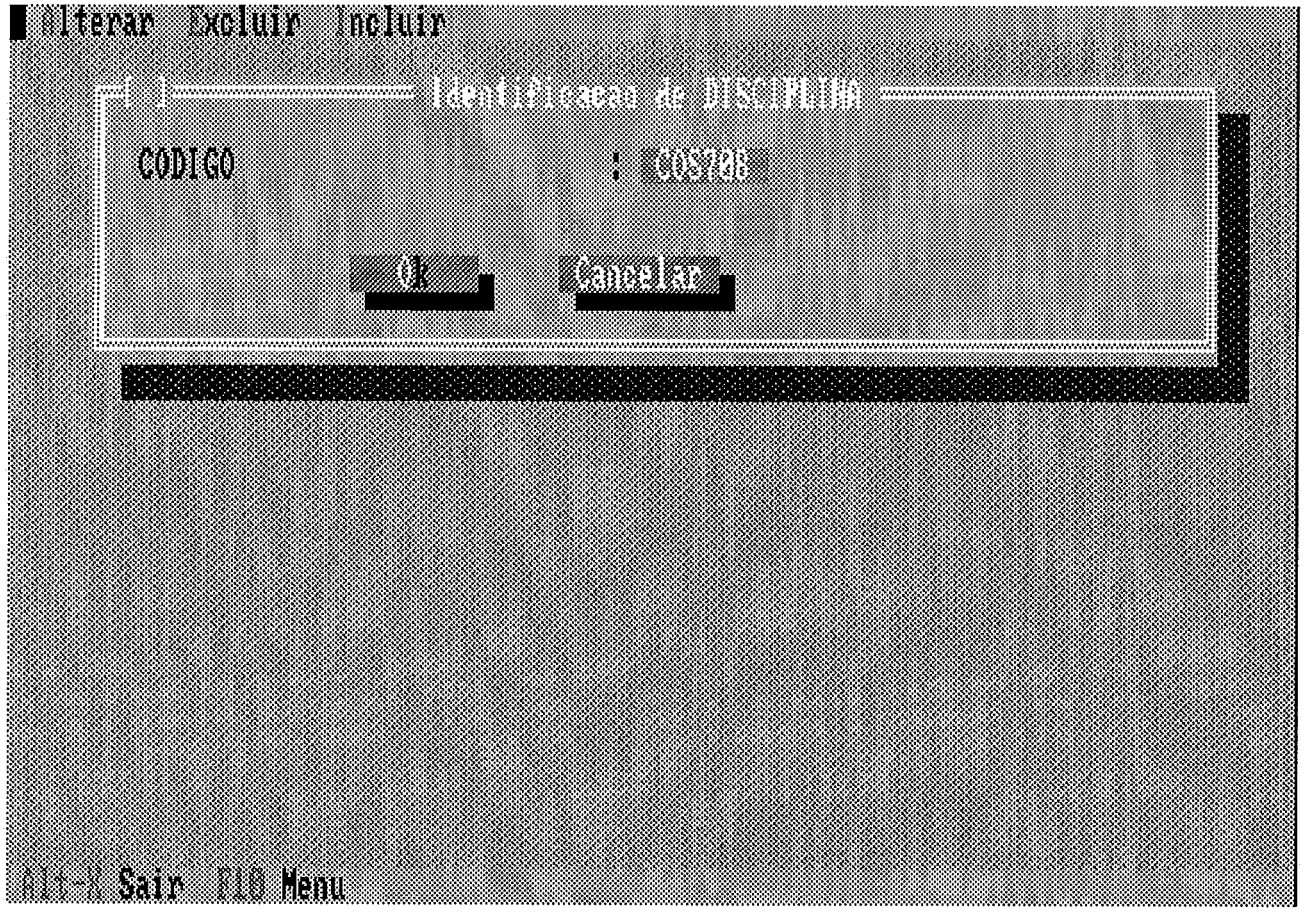
TRANSAÇÃO 4

O quarto exemplo de transação é a exclusão de uma instância de DISCIPLINA. Após a escolha da opção EXCLUSÃO, aparece a seguinte tela:



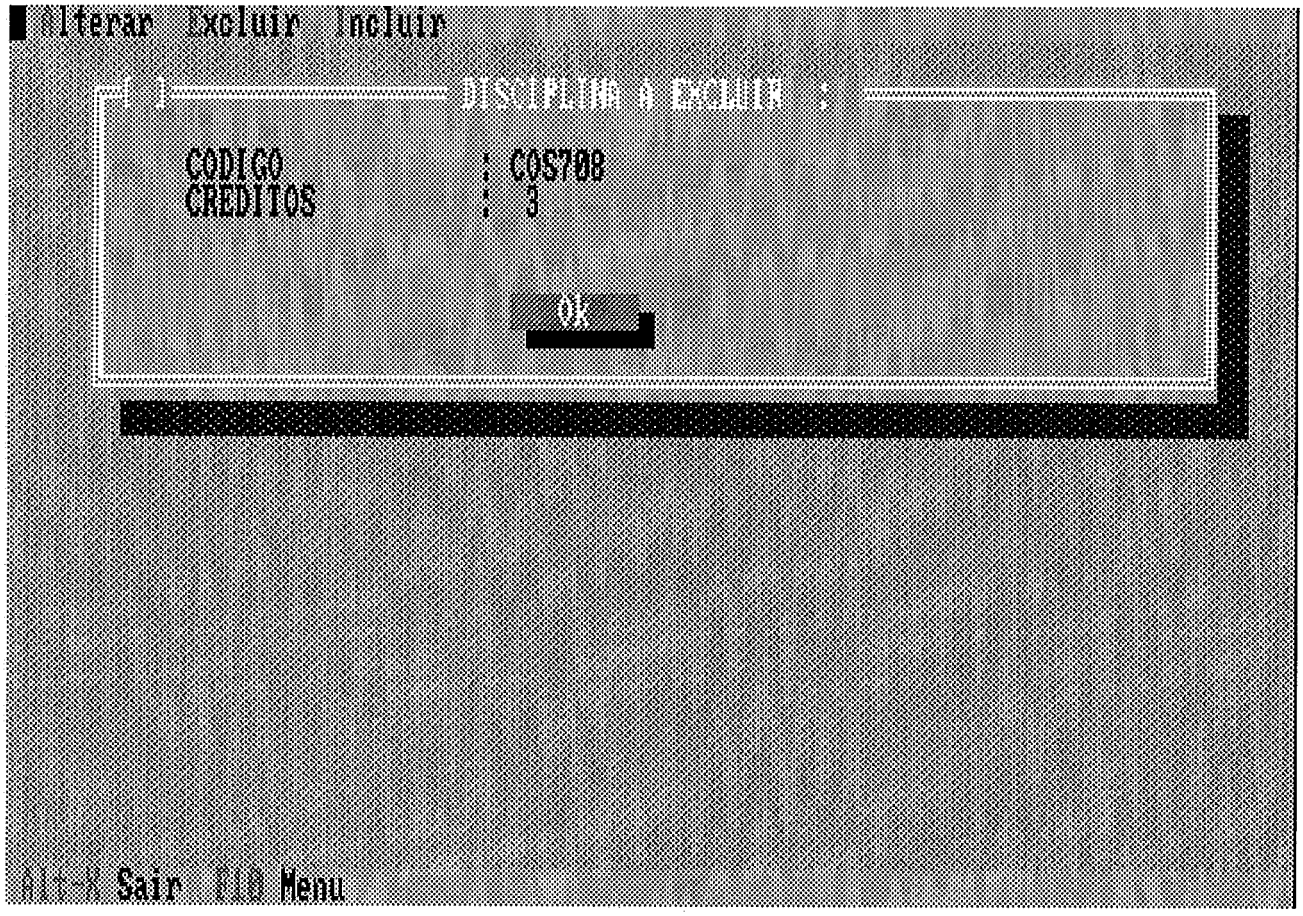
TELA **4.1**

Após a escolha da opção DISCIPLINA, aparece a tela para que se preencha o valor do atributo CÓDIGO, que vai ser usado na identificação da instância de DISCIPLINA que vai ser excluída:



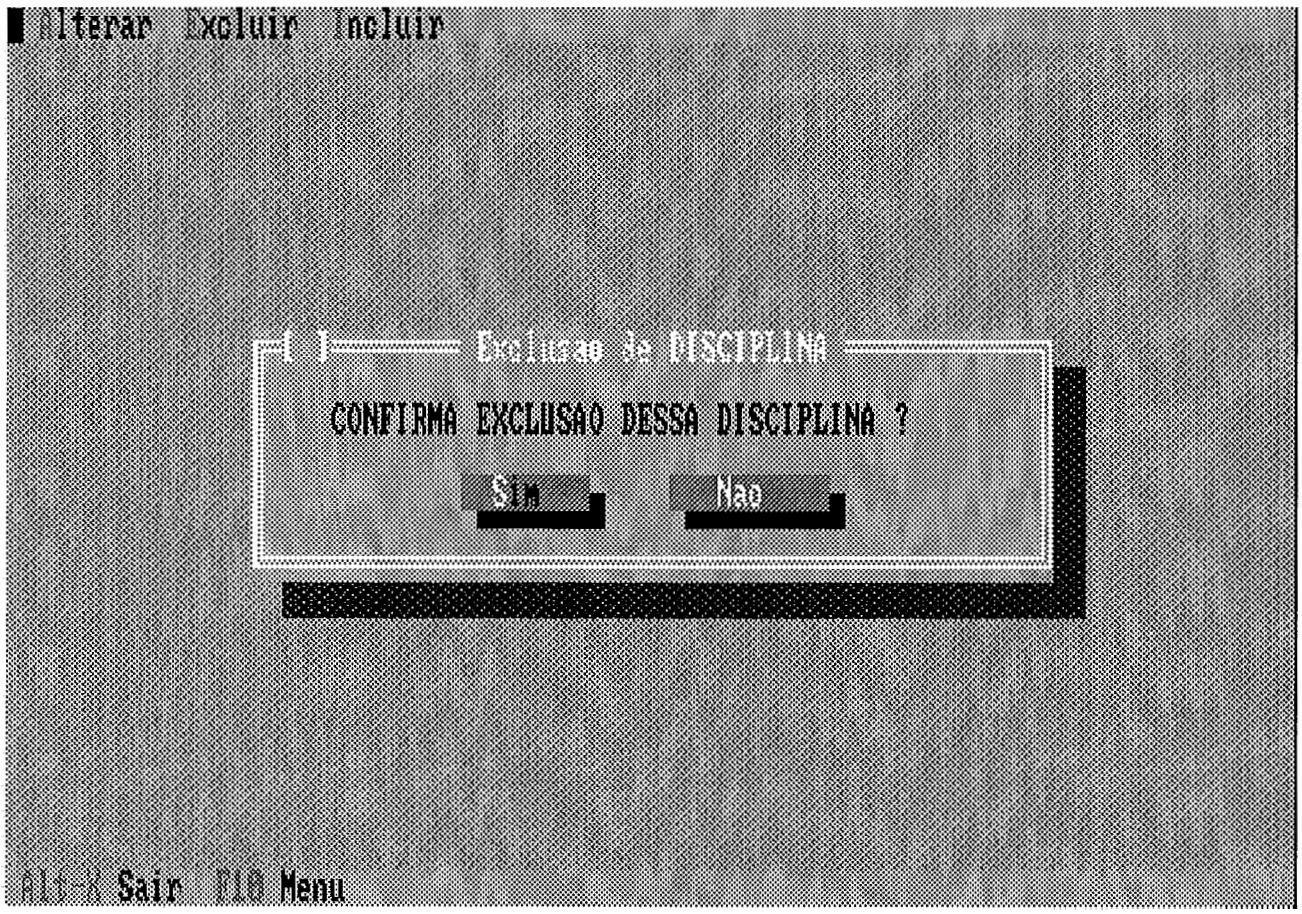
TELA 4.2

O usuário preenche o valor do atributo CÓDIGO. Depois, aparece uma tela mostrando os valores dos atributos da instância de DISCIPLINA que vai ser excluída:



TELA 4.3

A próxima tela serve para o usuário confirmar ou não a exclusão desta instância de DISCIPLINA:



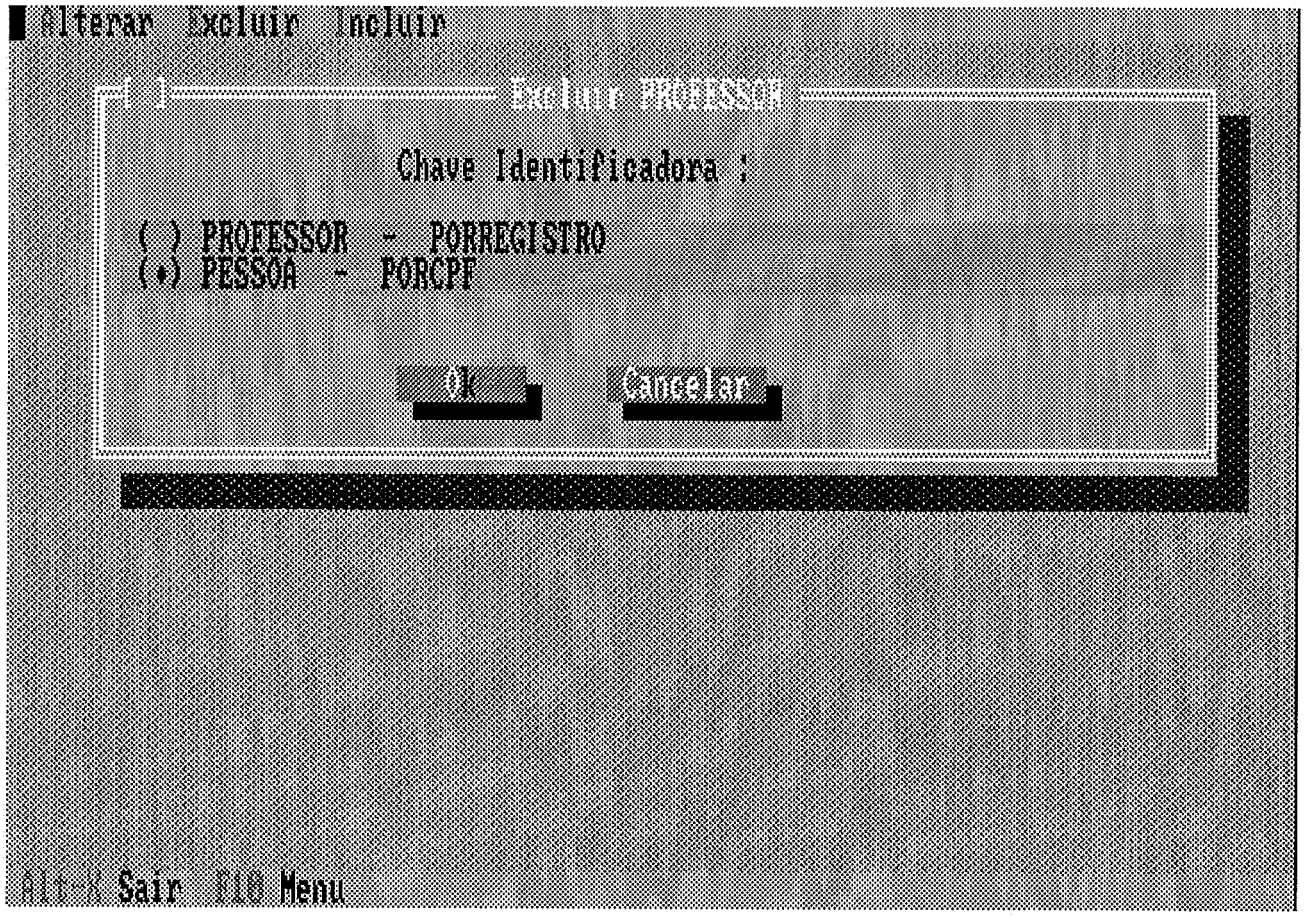
TELA 4.4

O usuário confirma a exclusão. A macrotransação de exclusão foi bem sucedida. Esta macrotransação foi composta somente da microtransação de exclusão da instância de DISCIPLINA.

TRANSAÇÃO 5

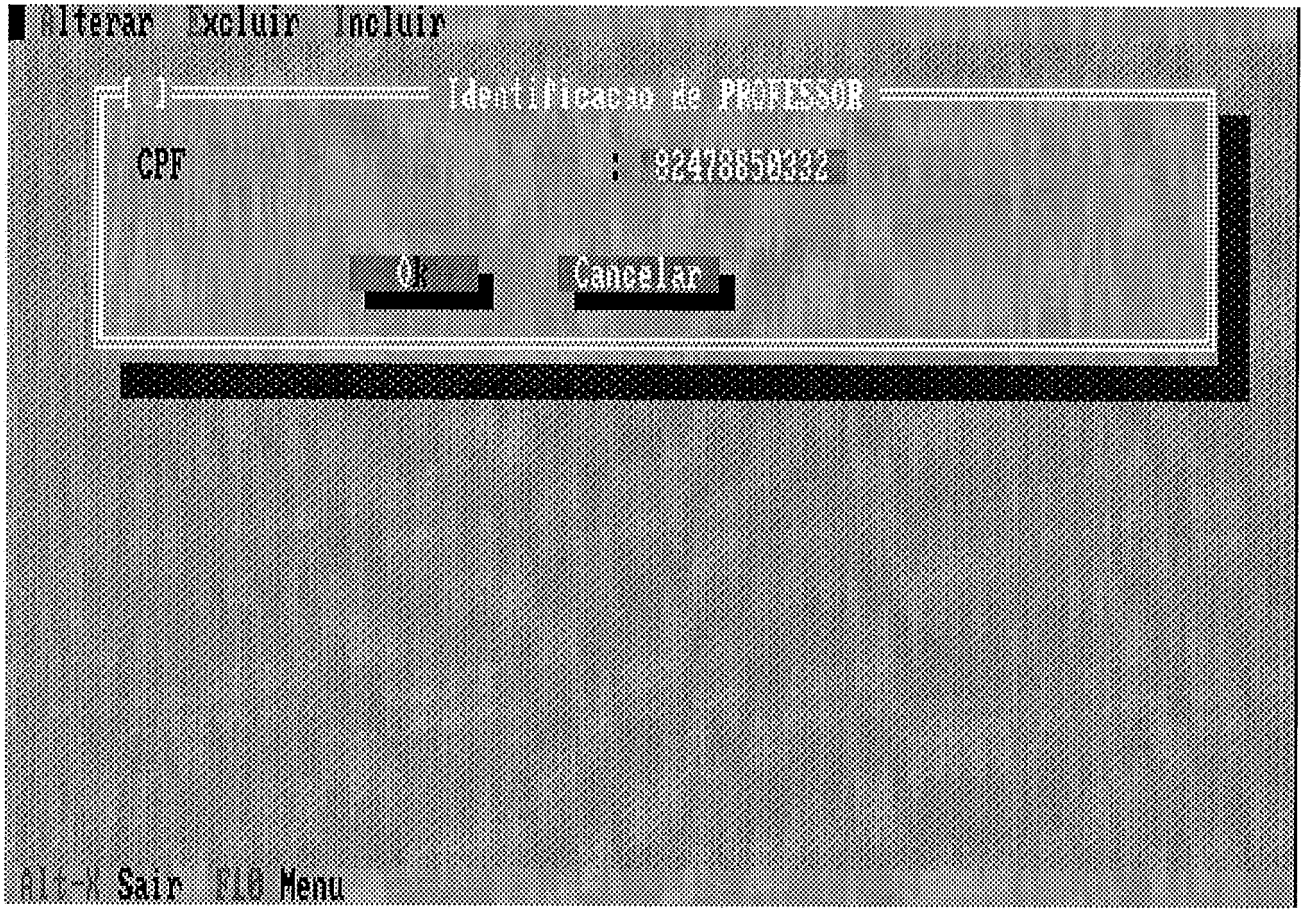
O quinto exemplo de transação é a exclusão de uma instância de PROFESSOR. Após a escolha da opção EXCLUSÃO, aparece a seguinte tela (VIDE TELA 4.1).

Após a escolha da opção PROFESSOR, aparece a tela para que se possa escolher a chave de acesso que vai ser utilizada para identificar a instância de PROFESSOR a ser excluída.



TELA 5.1

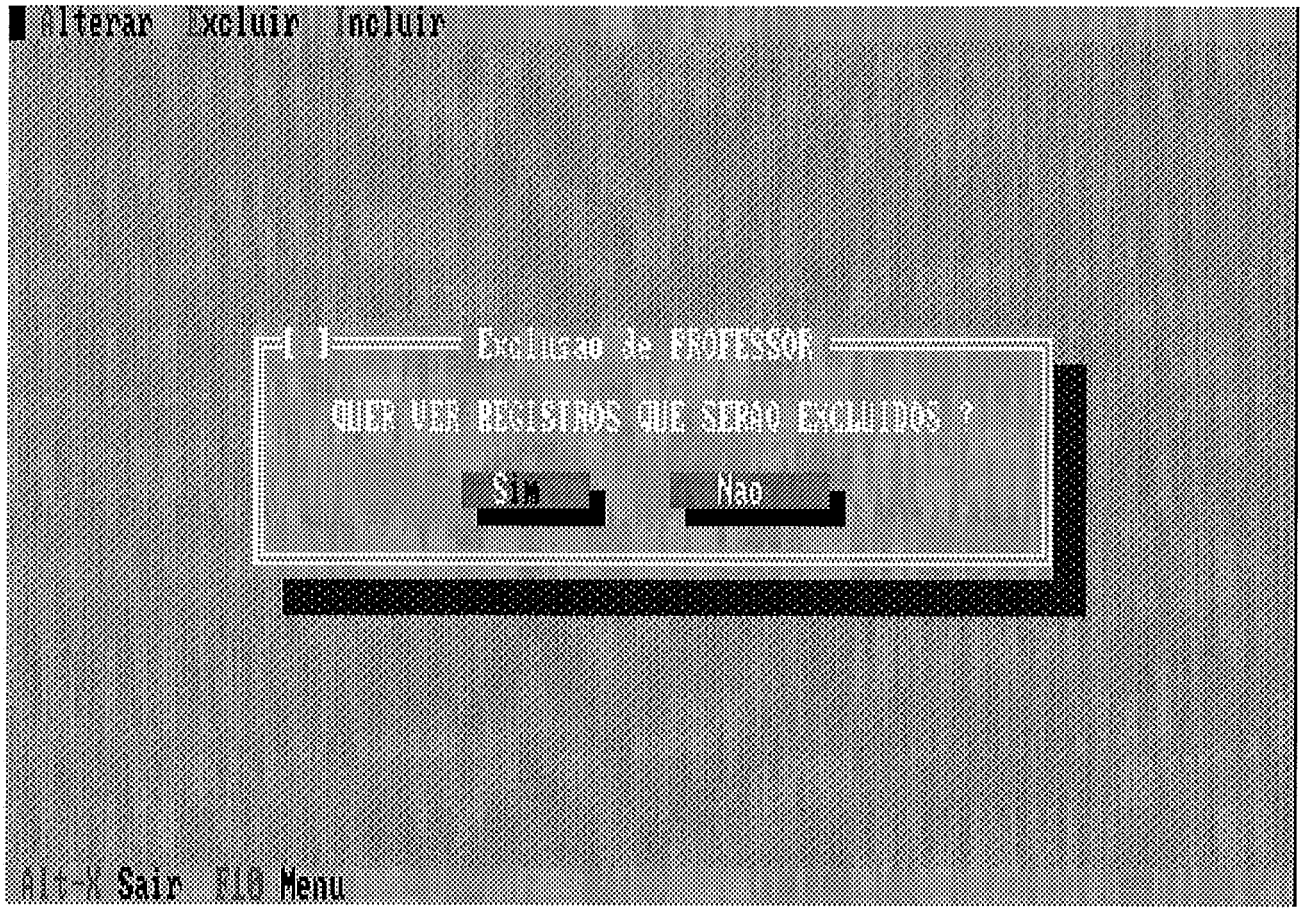
Após a escolha da chave de acesso PORCPF, aparece a tela para que se preencha o valor do atributo CPF:



TELA *5.2*

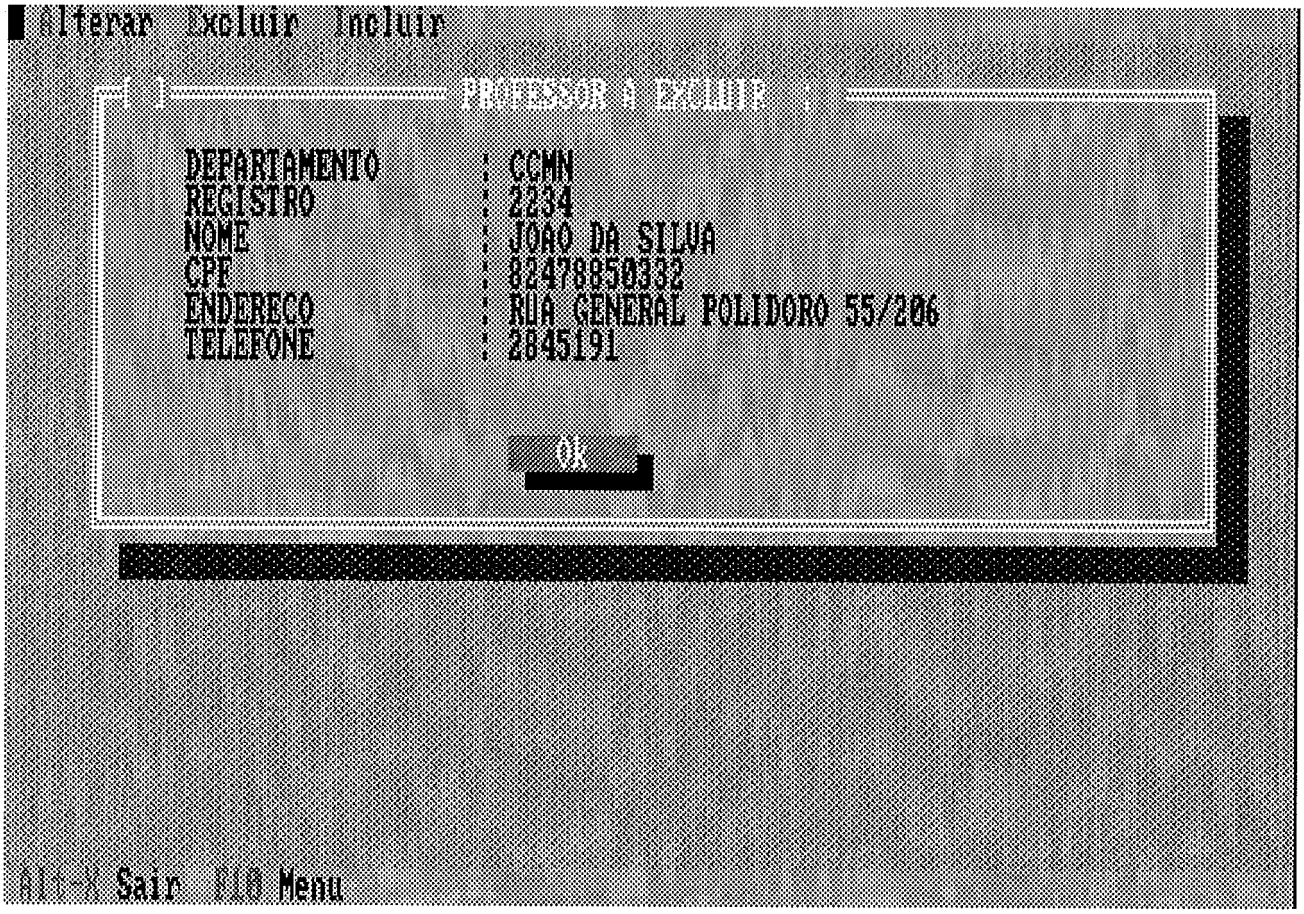
A exclusão desta instância de PROFESSOR vai acarretar uma propagação de exclusão, pois também deverá ser excluída a instância do relacionamento LECIONA que lhe corresponde.

A próxima tela pergunta se o usuário quer ver todas as instâncias que vão ser excluídas nesta macrotransação:



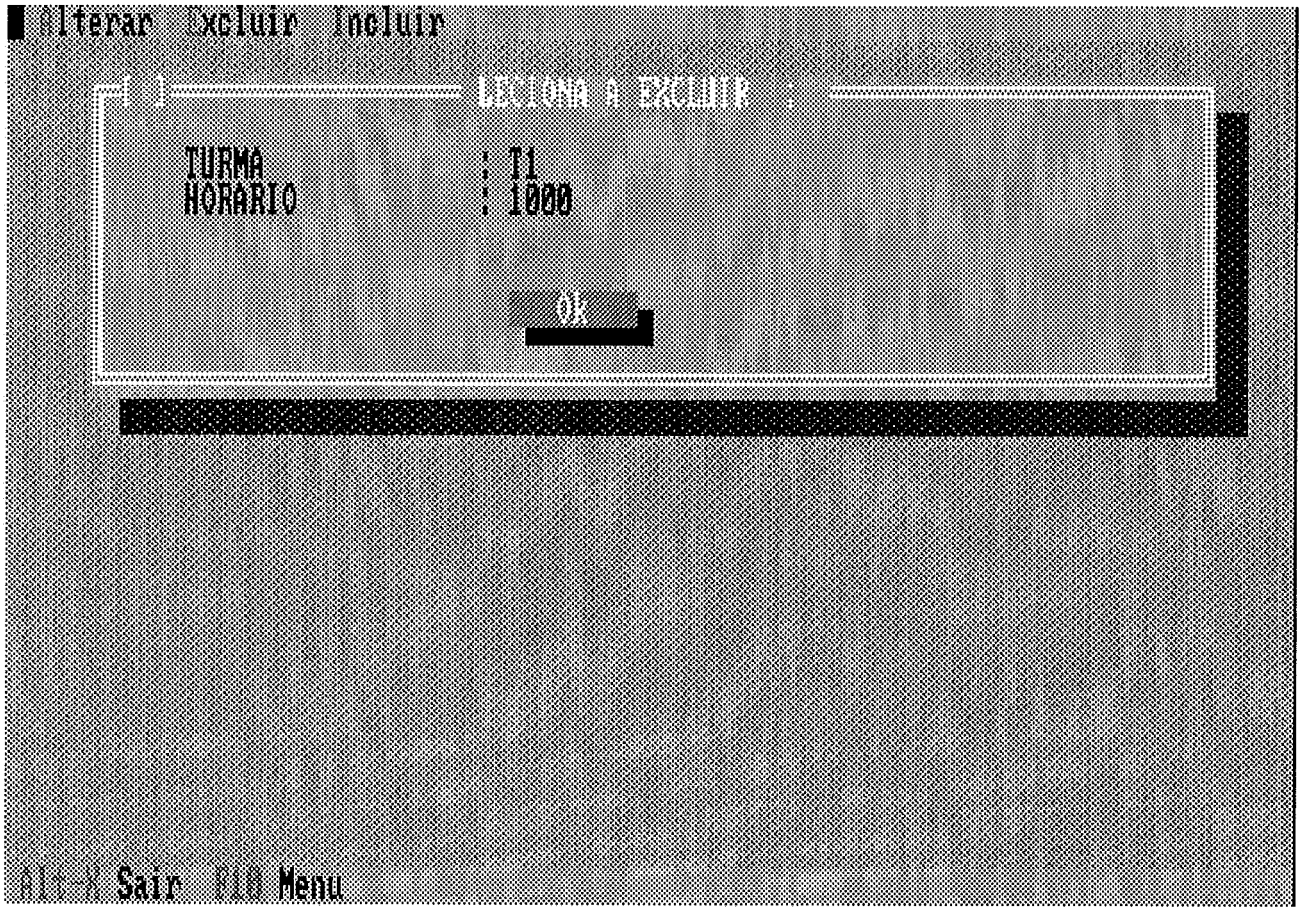
TELA *5.3*

Após o usuário ter escolhido a opção de ver todas as instâncias que vão ser excluídas, aparece uma tela mostrando os valores dos atributos da instância de PROFESSOR que vai ser excluída:



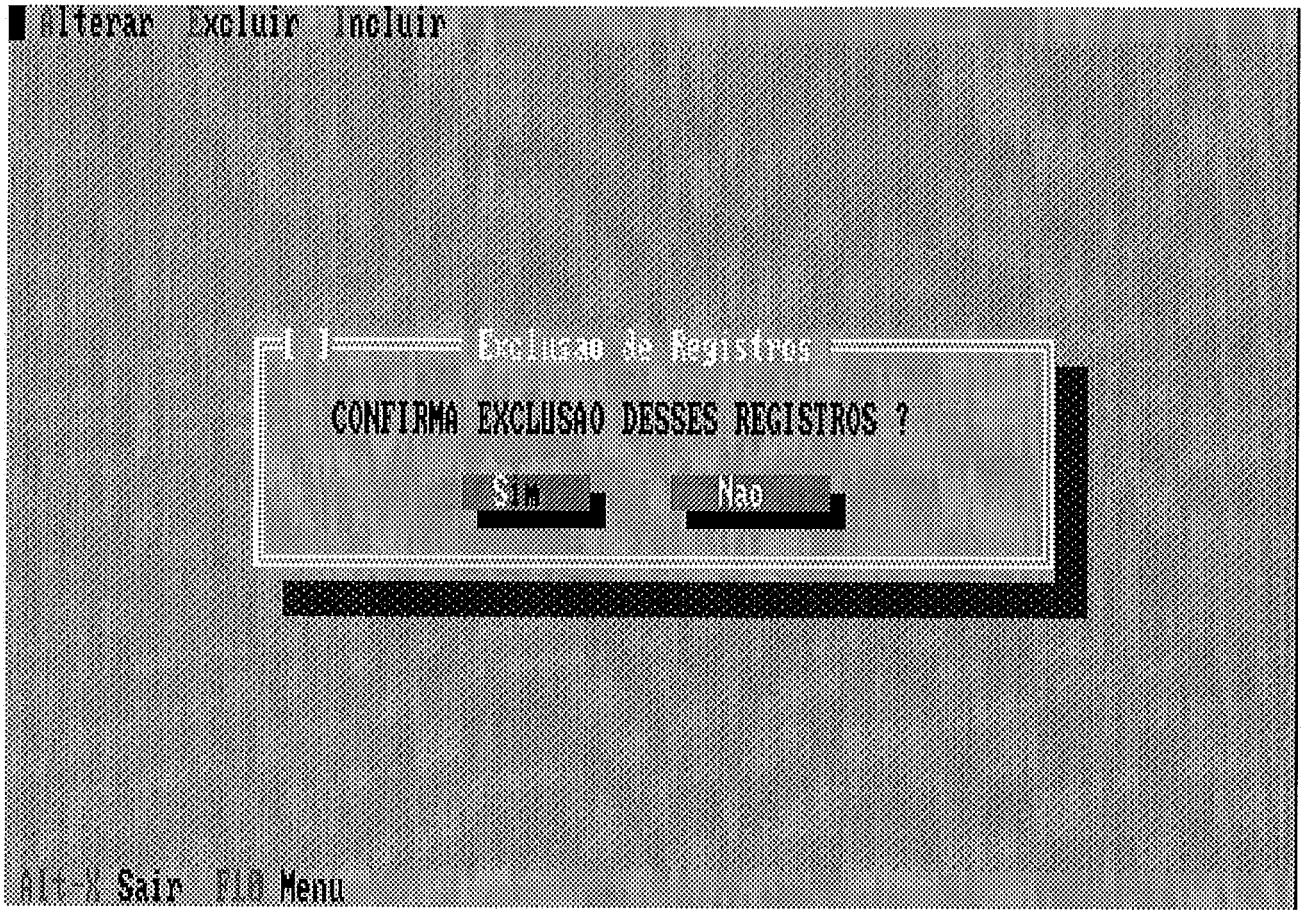
TELA 5.4

Depois, aparece uma tela mostrando os valores dos atributos da instância do relacionamento LECIONA que vai ser excluída:



TELA 5.5

A próxima tela serve para o usuário confirmar ou não a exclusão destas duas instâncias:



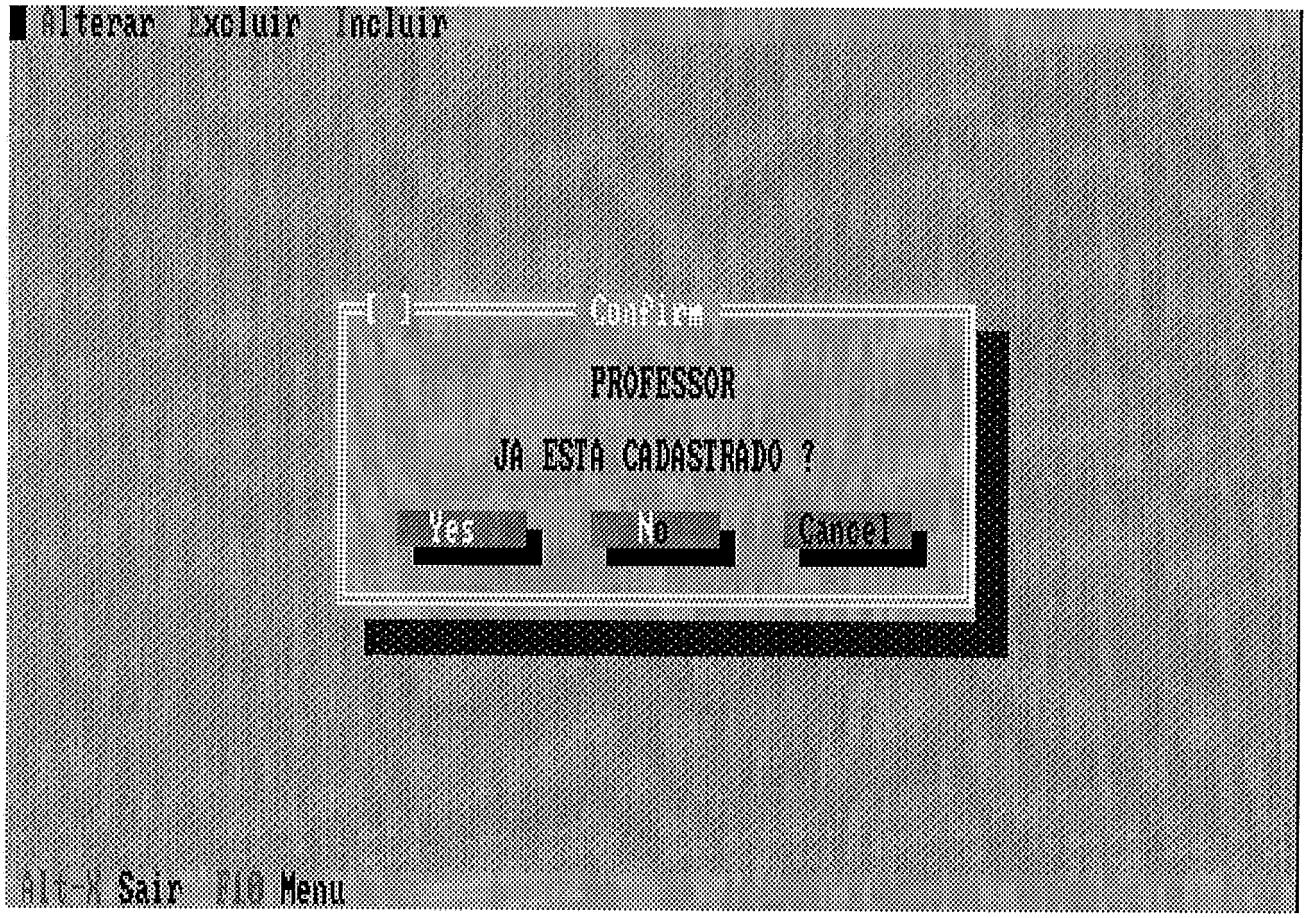
TELA 5.6

O usuário confirma a exclusão. A macrotransação de exclusão foi bem sucedida. Esta macrotransação foi composta da microtransação de exclusão da instância de PROFESSOR e da microtransação de exclusão da instância do relacionamento LECIONA.

TRANSAÇÃO 6

O sexto exemplo de transação é a inclusão de uma instância do relacionamento LECIONA. Após a escolha da opção INCLUSÃO, aparece a seguinte tela (VIDE TELA 1.1).

Após a escolha da opção LECIONA, aparece uma tela perguntando se a instância de PROFESSOR que vai tomar parte do relacionamento já existe ou precisa incluída:



TELA 6.1

O usuário responde que a instância de PROFESSOR precisa ser incluída. Aparece, então, a tela para o preenchimento dos valores dos atributos de PROFESSOR (VIDE TELA 2.2 e TELA 2.3).

Após a inclusão da instância de PROFESSOR, aparece uma tela perguntando se a instância de DISCIPLINA que vai tomar parte no relacionamento já existe ou precisa ser incluída:

TELA 6.2

O usuário responde que a instância já existe. Aparece, então, a tela para que se preencha o valor do atributo CÓDIGO, que vai ser usado na identificação da instância de DISCIPLINA que vai tomar parte no relacionamento (VIDE TELA 4.2).

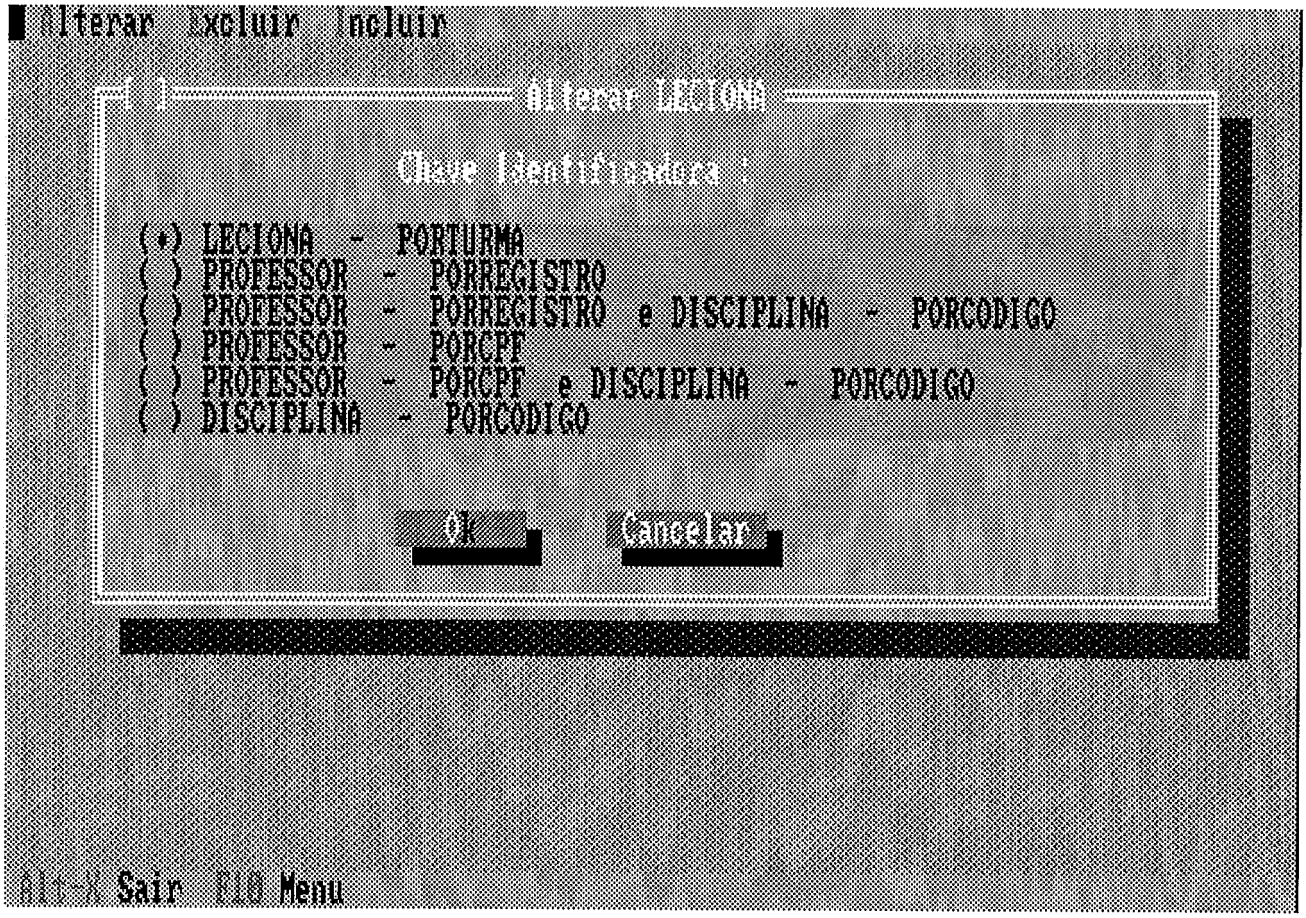
O usuário preenche o valor do atributo CÓDIGO. Aparece, então, a tela para preenchimento dos valores dos atributos do relacionamento LECIONA (VIDE TELA 2.6 e TELA 2.7).

A macrotransação de inclusão foi bem sucedida. Esta macrotransação foi composta das microtransações de inclusão da instância de PROFESSOR e inclusão da instância do relacionamento LECIONA.

TRANSAÇÃO 7

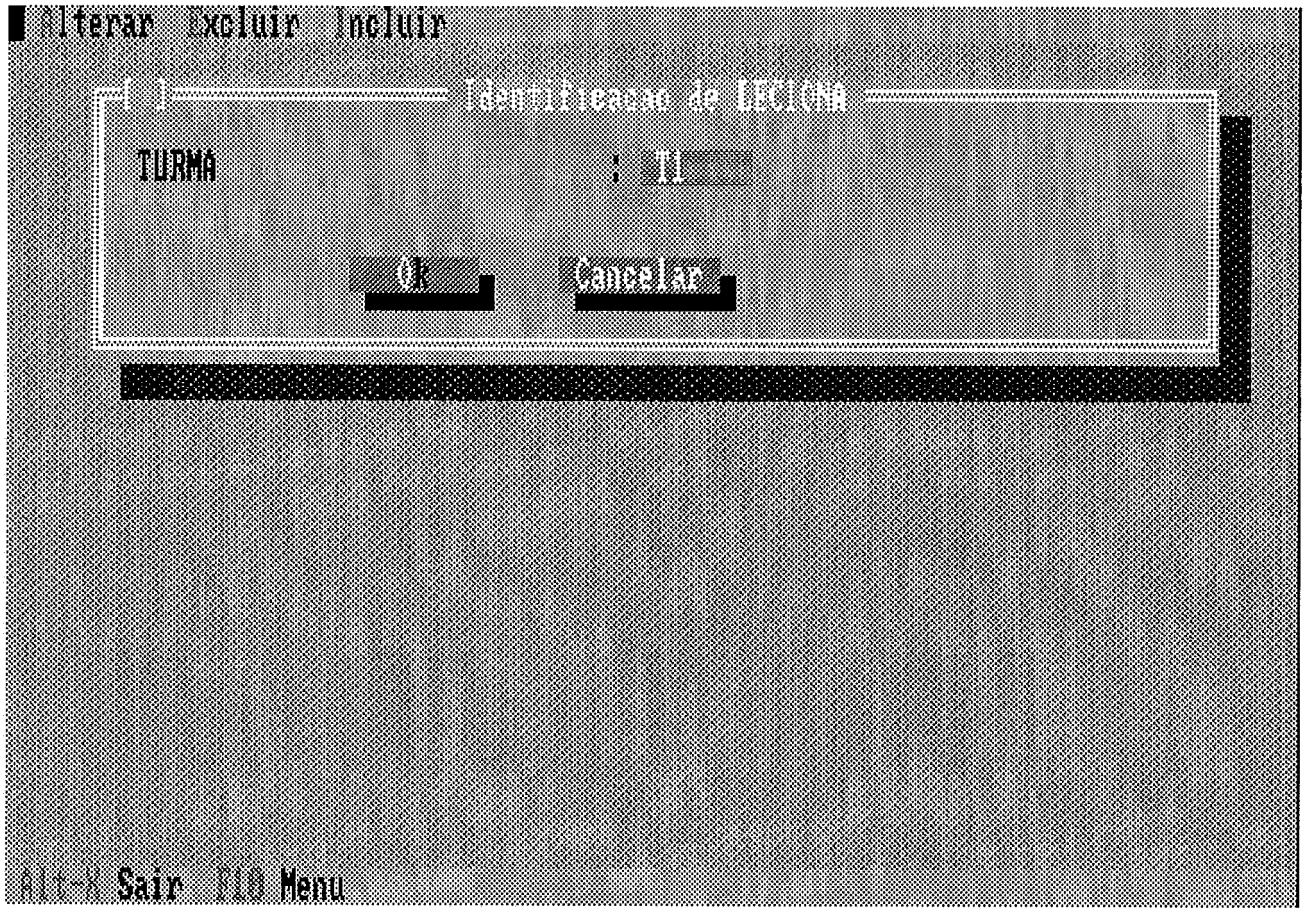
O sétimo exemplo de transação é a alteração de uma instância do relacionamento LECIONA. Após a escolha da opção ALTERAÇÃO, aparece a seguinte tela (VIDE TELA 3.1).

Após a escolha da opção LECIONA, aparece a tela para que se possa escolher a chave de acesso que vai ser utilizada para identificar a instância do relacionamento LECIONA que vai ser alterada.



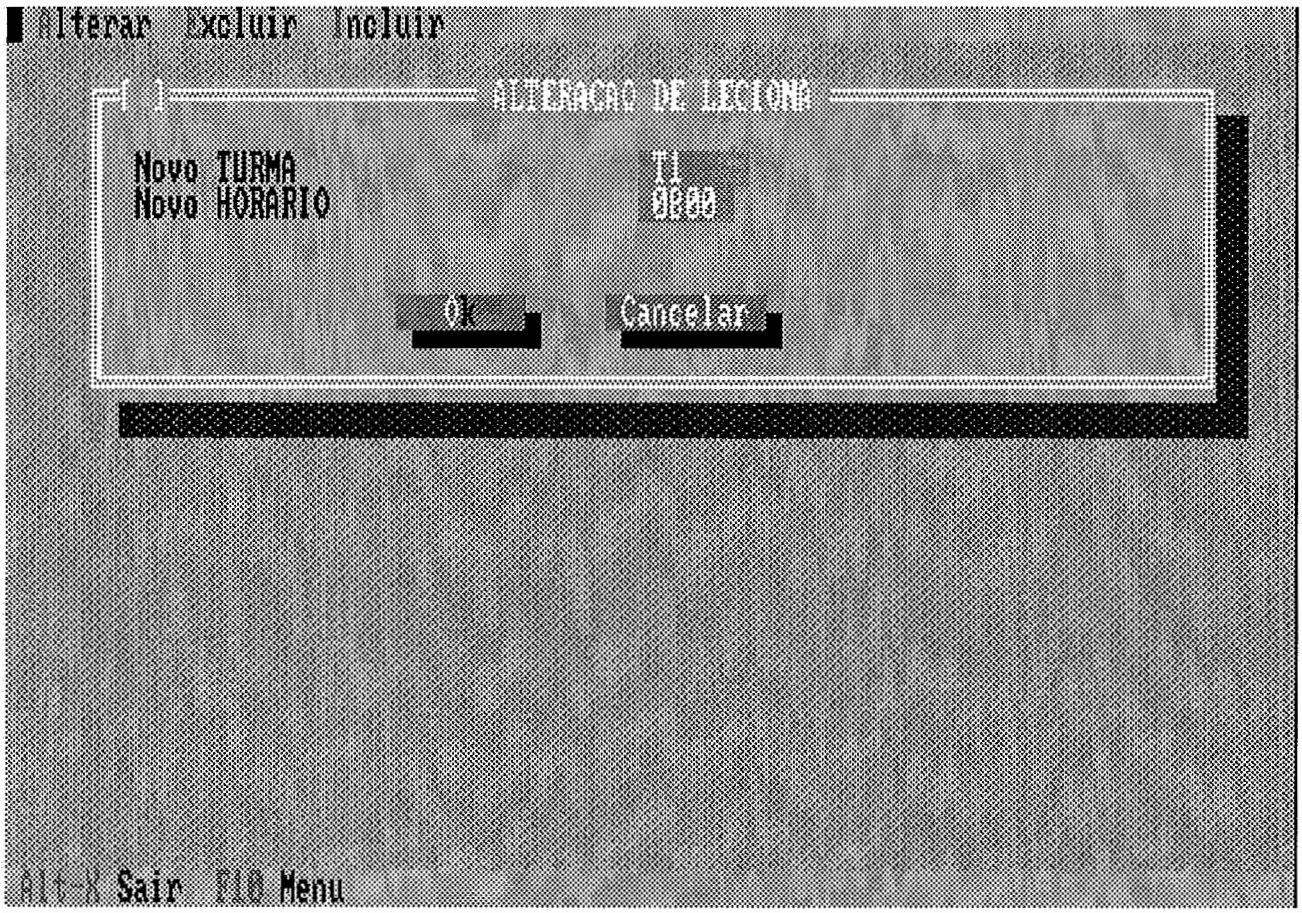
TELA 7.1

Após a escolha da chave de acesso PORTURMA, aparece a tela para que se preencha o valor do atributo TURMA:



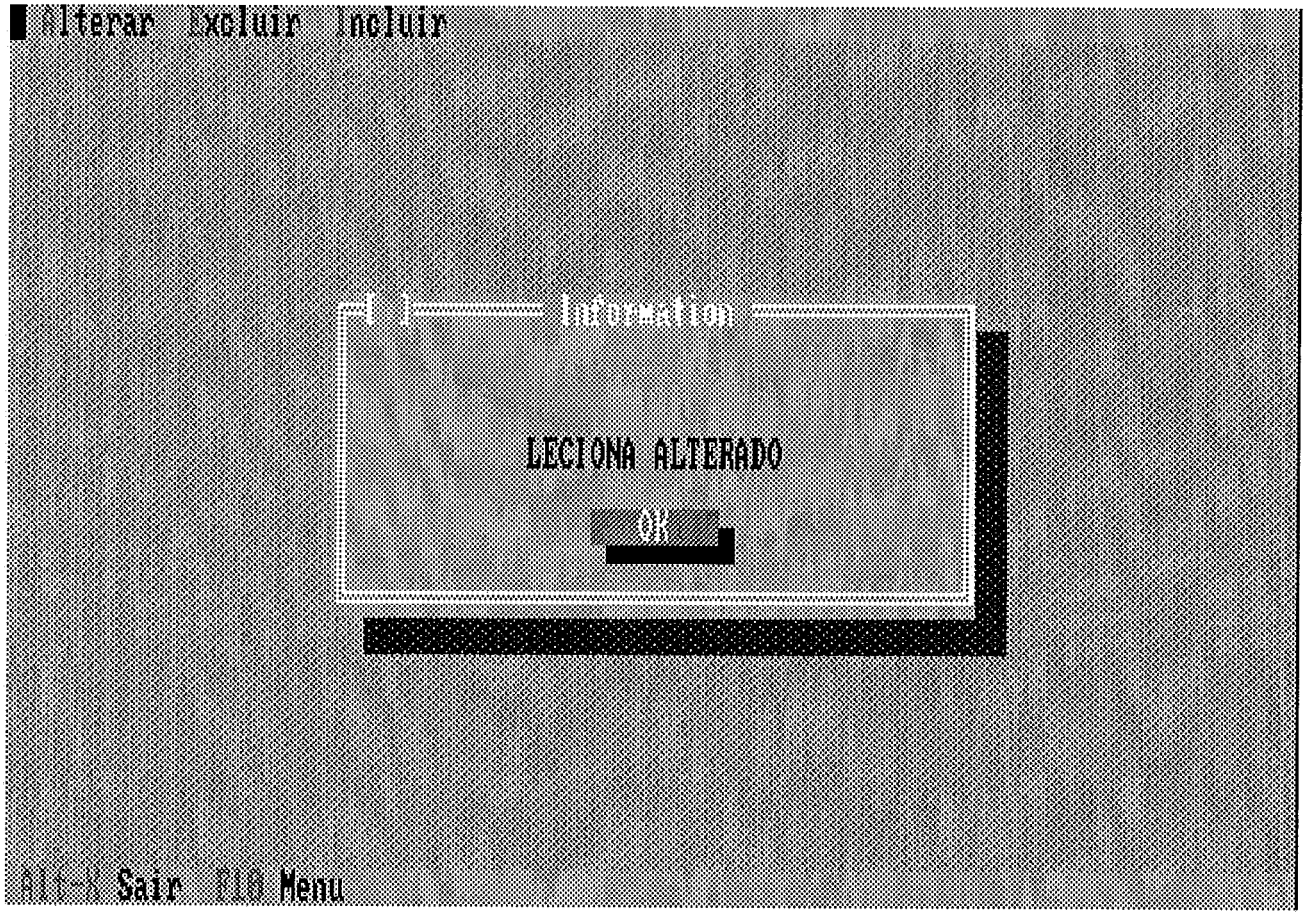
TELA 7.2

O usuário preenche o valor do atributo TURMA. Depois, aparece uma tela mostrando os valores atuais dos atributos da instância do relacionamento LECIONA que foi acessada:



TELA 7.3

O usuário altera os campos que quiser. Após a alteração da instância do relacionamento LECIONA, aparece uma mensagem informando que a transação foi bem sucedida:

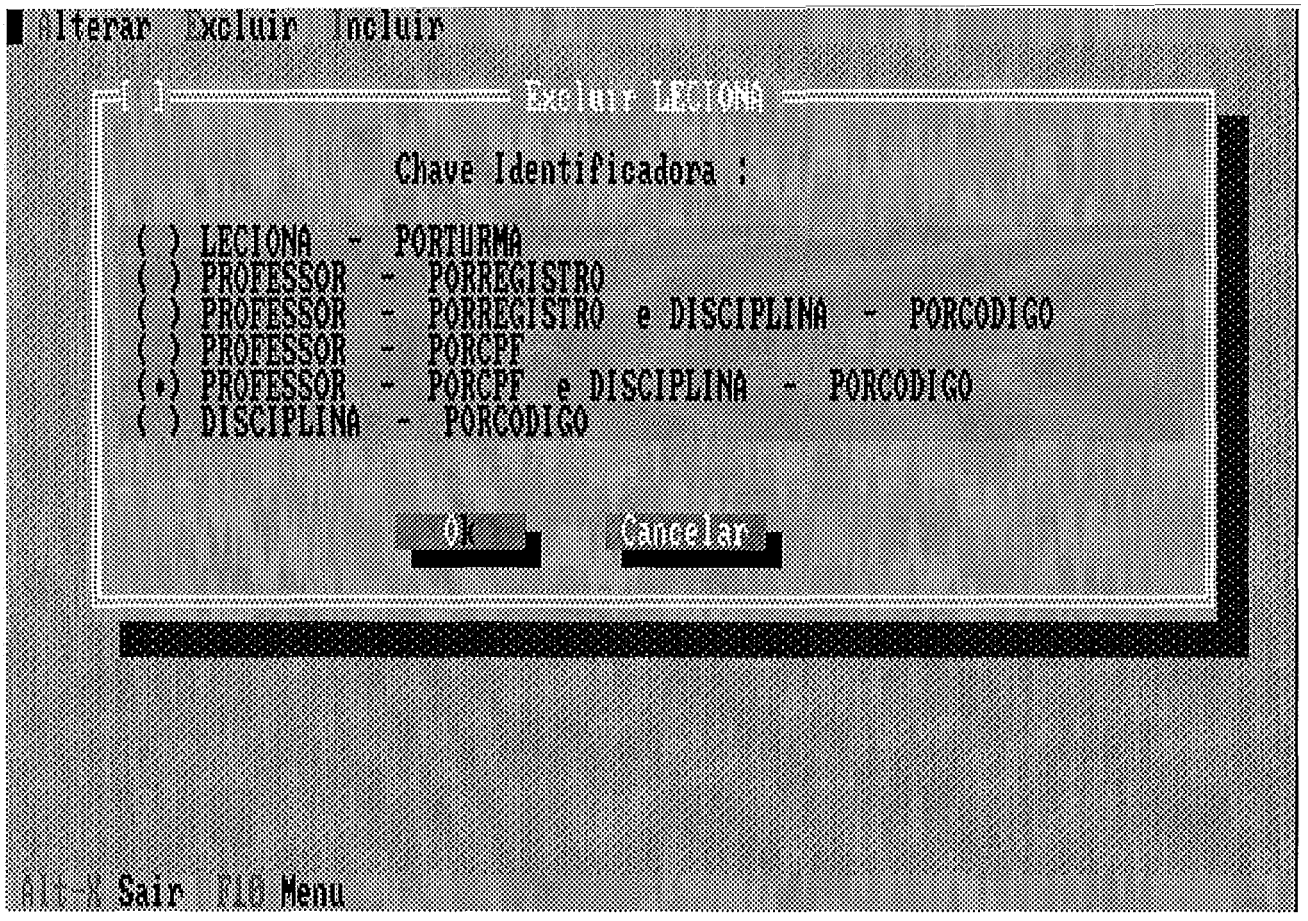


TELA 7.4

TRANSAÇÃO 8

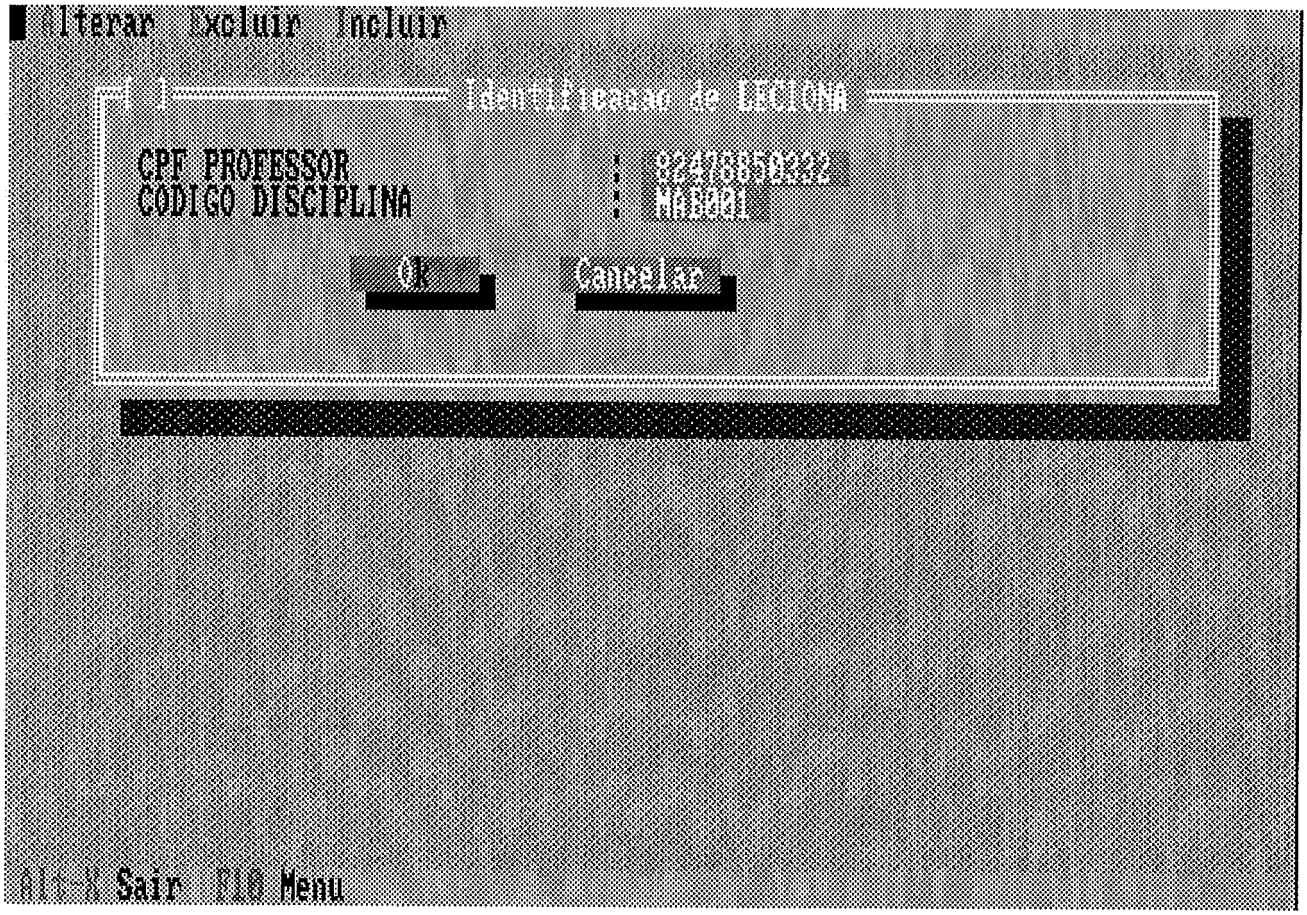
O oitavo exemplo de transação é a exclusão de uma instância do relacionamento LECIONA. Após a escolha da opção EXCLUSÃO, aparece a seguinte tela (VIDE TELA 4.1).

Após a escolha da opção LECIONA, aparece a tela para que se possa escolher a chave de acesso que vai ser utilizada para identificar a instância do relacionamento LECIONA que vai ser excluida.



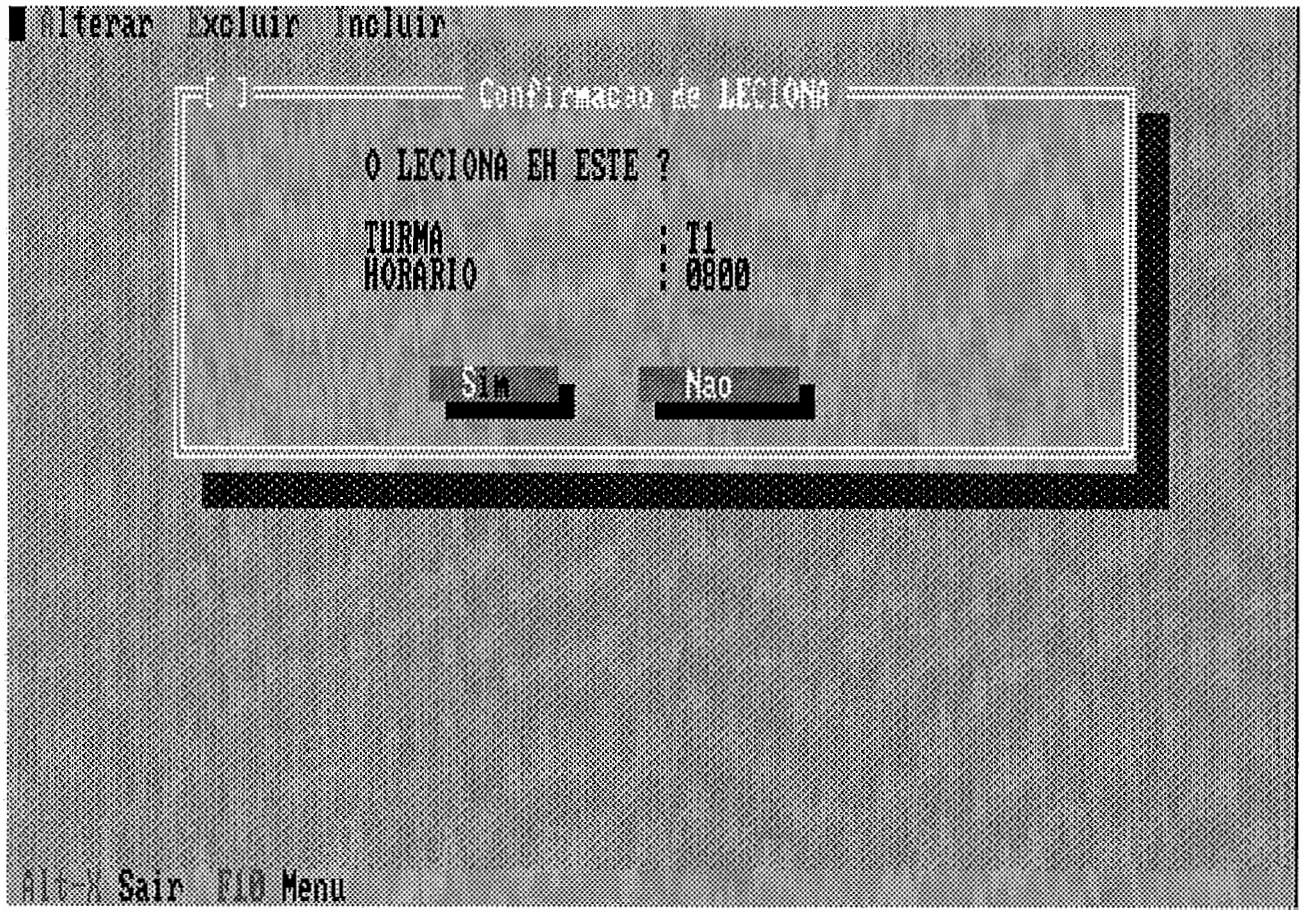
TELA 8.1

Após a escolha das chaves de acesso PORCPF e PORCODIGO, aparece a tela para que se preencha os valores dos atributos CPF e CODIGO:



TELA 8.2

O usuário preenche os valores dos atributos CPF e CÓDIGO. A utilização de chaves de acesso indiretas para se identificar a instância de um relacionamento faz com que se tenha de confirmar a instância acessada. Por isso, aparece a seguinte tela de confirmação:



TELA 8.3

O usuário confirma que esta é a instância do relacionamento que quer excluir. A exclusão desta instância do relacionamento LECIONA vai acarretar uma propagação de exclusão, pois também deverá ser excluída a instância de PROFESSOR que lhe corresponde.

A próxima tela pergunta se o usuário quer ver todas as instâncias que vão ser excluídas nesta macrotransação (VIDE TELA 5.4).

Após o usuário ter escolhido a opção de ver todas as instâncias que vão ser excluídas, aparece uma tela mostrando os valores dos atributos da instância de PROFESSOR que vai ser excluída (VIDE TELA 5.5).

Depois, aparece uma tela mostrando os valores dos atributos da instância do relacionamento LECIONA que vai ser excluída (VIDE TELA 5.6):

A próxima tela serve para o usuário confirmar ou não a exclusão destas duas instâncias (VIDE TELA 5.7).

O usuário confirma a exclusão, A macrotransação de exclusão foi bem sucedida. Esta macrotransação foi composta da microtransação de exclusão da instância de PROFESSOR e da microtransação de exclusão da instância do relacionamento LECIONA.