UNIVERSIDADE FEDERAL DA PARAIBA
CENTRO DE CIENCIAS E TECNOLOGIA
CURSO DE MESTRADO EM INFORMATICA

UMA ABORDAGEM HEURISTICA PARA A OTIMIZAÇAO SEMANTICA DE CONSULTAS A BANCOS DE DADOS DEDUTIVOS

José Alexandre Novaes Bicalho

CAMPINA GRANDE
MARÇO - 1991

JOSÉ ALEXANDRE NOVAES BICALHO

UMA ABORDAGEM HEURISTICA PARA A OTIMIZAÇAO SEMANTICA DE CONSULTAS A BANCOS DE DADOS DEDUTIVOS

Dissertação apresentada ao Curso de MESTRADO EM INFORMATICA da Universidade Federal da Paraíba, em cumprimento às exigências para obtenção do Grau de Mestre.

AREA DE CONCENTRAÇAO: CIENCIA DA COMPUTAÇÃO

MARCUS COSTA SAMPAIO
Orientador

CAMPINA GRANDE
MARÇO - 1991

B583a Bicalho, Jose Alexandre Novaes
Uma abordagem heuristica para a otimizacao semantica de consultas a bancos dedados dedutivos / Jose Alexandre Novaes Bicalho. - Campina Grande, 1991.
94 f.

1. Tese 2. Banco de Dados - Otimizador Semantico de Consultas I. Marcus Costa Sampaio, Dr. (orientador) II. Universidade Federal da Paraiba (PB) III. Título

CDU 004.63(043)

UMA ABORDAGEM HEURÍSTICA PARA A OTIMIZAÇÃO SEMÂNTICA
DE CONSULTAS A BANCOS DE DADOS DEDUTIVOS

JOSÉ ALEXANDRE NOVAES BICALHO

DISSERTAÇÃO APROVADA EM: 15/03/91

MARCUS COSTA SAMPAIO, M.Sc
Orientador

DECIO FONSECA, Dr.
Componente da Banca

STEPHANE PIERRE IURC, Dr.
Componente da Banca

JOSÉ HAMURABI N. DE MEDEIROS, M.Sc
Componente da Banca

CAMPINA GRANDE
MARÇO - 1991

A minha esposa Flávia,
e aos meus filhos Rodrigo
e Gustavo.

RESUMO

Este trabalho apresenta o projeto e a implementação de um otimizador **semântico de consultas**, desenvolvido em Prolog, para **bancos de dados dedutivos**. Ele se utiliza de informações semânticas expressas na forma de **restrições de integridade**, as quais, analisadas de acordo com heurísticas, determinam modificações na consulta original, visando obter uma melhora real de desempenho na sua execução sobre a base de fatos armazenada em um banco de dados relacional.

SUMARIO

I - Introdução

A utilização de otimização semântica de consultas a bancos de dados é relativamente recente. Ao contrário da otimização convencional, onde muitos algoritmos já foram apresentados e estão hoje implementados na maioria dos **sistemas de gerência de banco de dados** (SGBDs) comerciais, o desenvolvimento de idéias sobre otimização semântica está apenas começando.

Processos de otimização convencionais de consultas a bancos de dados relacionais baseiam-se principalmente em modificações sintáticas e em informações sobre o armazenamento das relações. Tais modificações incluem transformações algébricas e reordenação de operações, de acordo com informações como os métodos de acesso disponíveis e a cardinalidade das relações. Contudo, nesses processos, toda a semântica das informações armazenadas no banco de dados é desprezada.

Todo este conjunto de informações que representa a semântica do banco de dados pode ser visto como uma série de restrições de integridade (RIs) (Definição na seção 2.1 na Página 5). Estas RIs expressam afirmações verdadeiras sobre um determinado banco dados e estão divididas em categorias diferentes, umas relacionadas com o modelo de representação dos dados, como por exemplo o modelo relacional, chamadas de "dependências convencionais" e outras expressando informações genéricas sobre o banco de dados chamadas de "restrições semânticas".

Alguns artigos importantes como [KING81], que introduz a idéia de otimização semântica, e posteriormente [CHAK86],

[JARK84], [JARK86] e [SHEN87], apenas abrem a discussão para uma nova abordagem para o problema de otimização de consultas.

Em [KING81] alem de serem apresentadas os conceitos básicos sobre otimização semântica de consultas, são apresentadas uma série de regras de avaliação (heurísticas) que são utilizadas no processo de otimização, determinando a validade das modificações que são realizadas sobre a consulta original. Contudo, no seu trabalho é tratada apenas uma categoria de RIs (restrições semânticas), sendo desprezada toda a semântica inerente ao modelo relacional, que foi utilizado por ele.

[CHAK86] utilizando um banco de dados dedutivo (BDD), (Definição na seção 2.3 na Página 7) composto por fatos e regras de dedução ("visões"), apresenta um esquema de compilação semântica que visa incorporar, às regras de dedução, toda a informação semântica relevante, expressa nas RIs. Apesar do seu trabalho contemplar ambas as categorias de RIs, não é feita nenhuma análise heurística da validade efetiva dessas incorporações, como também não é apresentado o esquema do otimizador que utilizaria as informações semânticas incorporadas as regras de dedução.

Em [JARK84] e [JARK86], utilizando também BDDs, é apresentado um esquema de simplificação de consultas baseado em RIs da categoria "dependências convencionais", onde as informações semânticas inerentes ao modelo relacional são usadas para simplificar as consultas apresentadas pelo usuário. Contudo não são tratadas informações semânticas mais genéricas como as

expressas nas RIs da categoria "restrições semânticas" e também não é feita nenhuma consideração sobre a validade real (melhora no desempenho) das mudanças introduzidas na consulta.

[SHEN87] utiliza um algoritmo para realizar otimização semântica de consultas, utilizando algumas heurísticas como mecanismos de orientação do processo, contudo se restringe no tratamento de RIs da categoria "restrições semânticas".

No projeto ora relatado, são permitidas RIs tanto da categoria de restrições semânticas como da categoria de dependências convencionais, e todo o processo de modificação da consulta original é guiado por um conjunto de heurísticas que selecionam as modificações que conduzem a uma melhora real no desempenho da consulta. As heurísticas utilizadas foram apresentadas em [KING81] e expandidas para atender às novas categorias de RIs abordadas. Este trabalho descreve também o desenvolvimento e implementação de um protótipo para o projeto, sobre o qual foram realizados alguns testes preliminares, tendo sido obtidos bons resultados em vários casos de otimização.

No capítulo II, são apresentados alguns conceitos básicos utilizados na maioria dos trabalhos da área. É também apresentado um BDD exemplo que será utilizado durante todo o relato do trabalho.

No capítulo III, são apresentados alguns dos principais trabalhos na área de otimização semântica de consultas [KING81], [CHAK86] e [JARK84], visando situar este trabalho dentro do

universo de pesquisa.

No capítulo IV, é apresentada a estrutura geral do projeto, bem como um detalhamento dos módulos que o compõe. Para uma melhor compreensão do sistema, são utilizados vários exemplos durante a explicação do funcionamento de cada módulo.

O capítulo V, descreve as técnicas e os procedimentos usados na implementação de cada módulo do projeto, apresentando com detalhes os procedimentos correspondentes a cada heurística utilizada pelo otimizador.

No capítulo VI, inicialmente é feita uma análise do processo de otimização, e posteriormente, são apresentados vários exemplos de consultas submetidas ao otimizador, com suas respectivas avaliações de desempenho, permitindo uma análise das melhorias introduzidos pelo otimizador semântico.

No capítulo VII, são apresentadas as conclusões sobre a pesquisa e alguns trabalhos futuros que podem ser desenvolvidos com base no sistema ora apresentado.

Encerrando o trabalho, é apresentada a bibliografia dos principais trabalhos utilizados na pesquisa.

II - Conceituação Básica

Os itens apresentados a seguir têm por objetivo dar ao leitor uma visão geral dos conceitos utilizados.

2.1 - Restrições de integridade

As Restrições de Integridade (RIs) expressam a semântica do Banco de Dados e podem ser utilizadas no processo de otimização das consultas. Elas podem ser subdivididas em duas grandes categorias: as que expressam dependências convencionais e as restrições semânticas [ULLM82].

A primeira categoria engloba as dependências funcionais, dependências de chave, limites para valores (domínios de atributos) e integridades referenciais e representam informações inerentes ao modelo de representação dos dados , neste caso o modelo relacional.

Na segunda, estão as informações que representam o significado externo dos dados armazenados no Banco de Dados.

Como exemplo de uma RI da primeira categoria, representado uma dependência funcional, imaginemos um banco de dados sobre empregados onde a matrícula do empregado determina funcionalmente o nome do empregado, ou seja matricula -> nome. Esta dependência é uma restrição de integridade que garante que não podem existir dois empregados com a mesma matricula e com nomes diferentes.

Em outro exemplo, uma RI como: "Todo empregado que trabalha na presidência ganha mais de Cr$ 5000,00" representa uma restrição da segunda categoria garantindo que o salário de qualquer empregado que estiver lotado na presidência tem que ser sempre maior que Cr$ 5000,00.

RIs de ambas as categorias podem ser representadas em lógica de primeira ordem, como em [CHAK86] e [MINK88].

2.2 - Otimização Semântica baseada em heurísticas

A Otimização Semântica é um procedimento que realiza modificações em uma consulta apresentada pelo usuário, de acordo com informações semânticas sob a forma de RIs, gerando uma nova consulta, que produz o mesmo resultado, mas cujo desempenho (tempo de resposta) deve ser melhor que o da consulta na forma original [KING81].

As modificações são introduzidas de acordo com heurísticas, que determinam quando a aplicação de uma RI realmente traz alguma melhora para o desempenho da consulta.

Estas heurísticas podem ser vistas como regras de avaliação que são aplicadas sobre as modificações propostas pelas RIs, selecionando as modificações que efetivamente trazem alguma melhora no tempo de execução da consulta.

Os sistemas que realizam otimização semântica assumem que o banco de dados é consistente em relação as RIs, ou seja, as informações armazenadas no BD devem obedecer a

todas RIs. A função de verificação de consistência deve ser realizada por um outro sistema para que as RIs possam ser utilizadas no processo de otimização.

2.3 - Bancos de dados dedutivos

Bancos de dados dedutivos (BDDs) [GALL84] são sistemas que agrupam funções das áreas de Banco de Dados e de Inteligência Artificial. Eles se utilizam de toda a capacidade de armazenamento e recuperação de dados dos SGBDs, unida a mecanismos de inferencia. Tais sistemas são compostos basicamente de três partes. A primeira é o "banco de dados explícito" (BDE), onde são armazenados todos os fatos (axiomas). Neste trabalho o BDE é tratado como um conjunto de relações implementadas em um SGBD relacional.

A segunda parte é o "banco de dados implícito" (BDI), que é composto das regras de dedução utilizadas pelo mecanismo de inferência para dedução de novos fatos.

Por último, temos as RIs, onde está representada toda a semântica do BDD .

Uma consulta feita sobre um BDD dispara um mecanismo de dedução sobre as regras do BDI que aplicadas sobre os fatos do BDE produz os resultados da consulta.

2.4 - O BDD Exemplo

A linguagem Prolog (sintaxe de Edinburgh) aparece como uma forma de se representar o BDI e as RIs de um BDD

[LLOY85], como pode ser visto no BDD exemplo abaixo, que será utilizado em todo o corpo do trabalho. As variáveis estão representadas em letras maiúsculas e as constantes em letras minúsculas. Os "underscores" representam variáveis não identificadas.

Do BDE aparece apenas representado o esquema na forma usual de representação de esquemas relacionais já que os fatos que o compõe estarão armazenados efetivamente em um SGBD relacional. Os predicados que representam o BDE possuem um "E" no final do nome.

As RIs estão representadas em lógica de primeira ordem consistindo de implicações, possuindo antecedentes (<-A) e conseqüentes (C<--).

```
/* Banco de dados dedutivo sobre empregados

/* Esquema do banco de dados explícito - BDE

/* empE é uma relação do BDE que representa os empregados
/* composta pelos atributos matrícula, nome, salário e
/* número do departamento onde o empregado esta lotado.
empE(matr,nome,salario,dep).

/* deptE é uma relação do BDE representando os departamentos
/* que compõem a empresa composta pelo número do departamen-
/* to, função (nome) do departamento e a matrícula do chefe
/* do departamento.
deptE(dep,funcao,chefe).

/* especE é uma relação do BDE que representa as especiali-
```

```
/* dades dos  empregados  composta  pelos atributos nome  do
/* empregado e especialidade do empregado.
especE(nome,espec).

/* Banco de dados implícito - BDI

/* trabalha_dir_para é uma regra de dedução do BDI definindo
/* que um empregado com nome Nome1 trabalha diretamente para
/* um empregado  de nome Nome2  se o empregado de nome Nome1
/* trabalha  em um  departamento (Dept1) cujo  chefe (Matr2)
/* possui o nome Nome2.
trabalha_dir_para(Nome1,Nome2) :-
          empregado(_,Nome1,_,Dept1),
          dept(Dept1,_,Matr2),
          empregado(Matr2,Nome2,_,_).

/* mesmo_gerente é uma regra de dedução do BDI definindo que
/* um empregado cujo nome é Nome1 possui o  mesmo gerente de
/* um outro  empregado de  nome Nome2 se o empregado de nome
/* Nome1  trabalha  diretamente  para um  empregado de  nome
/* Gerente, se o empregado de nome Nome2 trabalha diretamen-
/* te para o mesmo empregado de Nome Gerente e  se os  nomes
/* dos empregados são diferentes.
mesmo_gerente(Nome1,Nome2) :-
          trabalha_dir_para(Nome1,Gerente),
          trabalha_dir_para(Nome2,Gerente),
          Nome1 \= Nome2.

/* empregado é uma regra de  dedução do BDI definindo que os
/* empregados da  empresa  são aqueles que estão armazenados
```

```
/* no BDE na relação EmpE.
empregado(Matr,Nome,Salario,Dep) :-
        empE(Matr,Nome,Salario,Dep).

/* dept é uma regra de  dedução do  BDI definindo que os de-
/* partamantos da empresa são aqueles que estão  armazenados
/* no BDE na relação deptE.
dept(Dep,Funcao,Chefe) :-
        deptE(Dep,Funcao,Chefe).

/* espec é uma  regra de  dedução do  BDI definindo  que  as
/* especialidades dos  empregados da empresa são aquelas que
/* estão armazenadas no BDE na relação especE.
espec(Nome,Espec) :-
        especE(Nome,Espec).
/* Restrições de integridade - RIs

    /* Categoria: Dependências convencionais

        /* Dependências funcionais

        /* RI1 especifica a dependência funcional
        /*          nome -> matrícula.
        /* Portanto se  existirem  dois  empregados com  o
        /* mesmo  nome (Nome)  suas  matrículas  devem ser
        /* iguais (Matr1 = Matr2).
        RI1 - restricao_de_integridade([
                (Matr1=Matr2)<--,
                <-empE(Matr1,Nome,_,_),
                <-empE(Matr2,Nome,_,_)]).
```

```
/* RI2 especifica a dependência funcional
/* matrícula do chefe -> numero do departamento.
/* Portanto se existirem dois departamentos  com o
/* mesmo  chefe (Chefe) o  número do  departamento
/* deve ser o mesmo (Dep1 = Dep2).
RI2 - restricao_de_integridade([
          (Dep1=Dep2)<--,
          <-deptE(Dep1,_,Chefe),
          <-deptE(Dep2,_,Chefe)]).

/* Dependências de chave */

/* RI3 especifica a dependência de chave
/*     matrícula -> nome, departamento, salário.
/* Portanto  se  existirem  dois empregados  com a
/* mesma  matrícula (Matr)  o nome, o departamento
/* e o salário devem ser iguais.
RI3 - restricao_de_integridade([
          (Nome1=Nome2,Dep1=Dep2,Sal1=Sal2)<--,
          <-empE(Matr,Nome1,Sal1,Dep1),
          <-empE(Matr,Nome2,Sal2,Dep2)]).

/* RI4 especifica a dependência de chave
/*     numero departamento -> funcao, chefe.
/* Portanto se existirem dois  departamentos com o
/* mesmo número (Dep) a função e o chefe devem ser
/* iguais.
RI4 - restricao_de_integridade([
```

```
            (Func1=Func2,Chefe1=Chefe2)<--,
            <-deptE(Dep,Func1,Chefe1),
            <-deptE(Dep,Func2,Chefe2)]).

/* Integridades referenciais */

/* RI5 especifica uma integridade referencial
/* entre o número do departamento na relação DeptE
/* e o número do departamento na relação empE, ga-
/* rantindo que todos os empregados estão lotados
/* em um departamento.
RI5 - restricao_de_integridade([
            deptE(Dep,_,_)<--,
            <-empE(_,_,_,Dep)]).

/* RI6 especifica uma integridade referencial
/* a matrícula do chefe de um departamento na re-
/* lação DeptE e a matrícula do empregado na rela-
/* ção empE, garantindo que todos os chefes de de-
/* partamentos são empregados.
RI6 - restricao_de_integridade([
            empE(Matr,_,_,_)<--,
            <-deptE(_,_,Matr)]).

/* Limites para valores */

/* RI7 especifica os limites para os salários de
/* um empregado. O salário de todos os empregados
/* devem ser maiores que 1000 e menores que 10000.
RI7 - restricao_de_integridade([
```

```
                Sal>1000<--,
                Sal<10000<--,
                <-empE(_,_,Sal,_)]).

/* Categoria: Restrições semânticas */

    /* RI8 é uma restrição semântica especificando que
    /* o empregado cujo salário (Sal) seja maior ou
    /* igual a 8000 está lotado em um departamento
    /* (Dep) cuja função é a presidência e possui a
    /* a especialidade de jogar golf.
    RI8 - restricao_de_integridade([
                deptE(Dep,presidencia,_)<--,
                especE(Nome,golf)<--,
                <-empE(_,Nome,Sal,Dep),
                <-Sal >= 8000]).

    /* RI9 é uma restrição semântica especificando que
    /* todo empregado que está lotado no departamento
    /* de número 4 recebe um salário > 8000.
    RI9 - restricao_de_integridade([
                Sal >= 8000<--,
                <-empE(_,_,Sal,04)]).
```

III - Principais pesquisas na área

A seguir, serão apresentados alguns projetos na área de otimização semântica de consultas, visando situar a nossa pesquisa dentro de um contexto global.

3.1 - QUIST: Um otimizador semântico de consultas para bancos de dados relacionais

O projeto QUIST (QUery Improvement through Semantic Transformation) [KING81] é um otimizador semântico de consultas para bancos de dados relacionais que se utiliza de "conhecimento" expresso em restrições de integridade da categoria restrições semânticas, para realizar modificações sobre a consulta original submetida pelo usuário.

O otimizador realiza transformações sobre um subconjunto das operações relacionais composto de seleções, projeções e junções. A linguagem de consulta utilizada pelos usuários é formada por expressões booleanas envolvendo seleções sobre os atributos do banco de dados (BD) e por uma lista dos atributos que se deseja obter como resposta. A eliminação dos operadores de junção da consulta é possível, pois existe uma única junção entre os pares de relações do BD (restrição imposta pelo projeto) e as relações envolvidas na consulta são aquelas cujos atributos apareceram na consulta.

O processo de otimização se baseia no paradigma da inteligência artificial "planejamento-geração-teste". O

nível de planejamento é utilizado na escolha das relações "alvo" para as modificações propostas pelas RIs. Esta escolha é baseada em heurísticas que determinam as relações alvo, bem como as relações "não-alvo", que são aquelas que não precisam sofrer novas modificações na consulta.

O nível de geração é representado pela formulação das novas consultas, a partir das modificações propostas pelas RIs. O nível de teste é realizado através de uma avaliação de custo das diversas consultas geradas a partir da consulta original.

Um fluxo geral do processo de otimização pode ser visto abaixo:

Início;

Determina as relações alvo e não-alvo, de acordo com as heurísticas;

Enquanto existirem relações alvo faça:

Seleciona as RIs relevantes, visando as relações alvo determinadas no passo anterior;

Analisa a aplicabilidade de cada RI, verificando se as suas precondições são satisfeitas pela consulta;

Se existirem RIs aplicáveis

Introduz as modificações impostas pelas RIs selecionadas, gerando várias consultas

alternativas;

Senão

Saia do LOOP;

Fim Se;

Reavalia as relações alvo;

Fim Enquanto

Realiza uma avaliação de custo sobre as consultas alternativas geradas no loop anterior;

Seleciona a consulta de menor custo para ser executada.

Fim.

As heurísticas utilizadas na determinação das relações alvo têm em vista o modelo utilizado para avaliação de custos realizada no nível de teste. Ou seja, as heurísticas foram feitas de acordo com o processo de avaliação que é executado posteriormente. Isto indica que a utilização das heurísticas deve trazer uma melhora no desempenho da execução das novas consultas produzidas. As heurísticas que determinam relações alvo são:

H1 - Introdução de índice. Uma relação que originalmente exista na consulta deve ser alvo para a inclusão de uma nova restrição sobre um atributo, pelo qual exista um índice. Neste caso, o objetivo é trocar uma pesquisa sequencial sobre uma relação por uma pesquisa indexada pelo atributo chave.

H2 - Introdução de junção. Uma relação deve ser considerada alvo, caso exista entre ela e uma outra grande relação da consulta, um índice agrupado ("cluster index"), mesmo que a primeira relação não faça parte da consulta original, obrigando a introdução desta nova junção. O objetivo, neste caso, é o mesmo buscado com a aplicação da heurística anterior.

H3 - Redução de pesquisa. Uma relação que está envolvida em uma junção com uma outra relação que já está suficientemente restringida na consulta, deve ser determinada como alvo. Neste caso, o objetivo é gerar uma relação temporária de cardinalidade menor, para ser utilizada na junção.

H4 - Eliminação de junção. Um relação da consulta que possua apenas uma junção com uma outra relação e que nenhum dos seus atributos contribuam para a resposta da consulta, deve ser considerada alvo, visando a sua eliminação da consulta.

Heurísticas que determinam relações não-alvo:

H5 - Relações que não fazem parte da consulta original não devem ser consideradas alvo, com exceção para o caso em H2.

H6 - Uma relação que já está na consulta, não deve ser considerada alvo, caso possua um índice agrupado com

uma outra relação já restringida na consulta.

H7 - Uma relação que esteja fortemente restringida na consulta sobre um atributo que possua um índice agrupado não deve ser considerada alvo para novas restrições.

Para ilustrar o funcionamento do QUIST, veremos a seguir um exemplo. O esquema do BD exemplo é:

BARCOS(NomeBarco, Dono, TipoBarco, Capacidade, Registro)
PORTOS(NomePorto, Pais, Facilidade)
CARGAS(Barco, Destino, TipoCarga, Quantidade, Valor,Seguro)
DONOS(NomeDono, Localizacao, Negocio)
APOLICES(Apolice, Segurador, Cobertura)
SEGURADORAS(NomeSegurador, Capitalizacao)

Com as seguintes cardinalidades:

Card(BARCOS) = 20.000
Card(PORTOS) = 1.000
Card(CARGAS) = 25.000
Card(DONOS) = 1.000
Card(APOLICES) = 25.000
Card(SEGURADORAS) = 500

As junções implicitas que identificam os caminhos lógicos de acesso nas consultas são:

DONOS.NomeDono = BARCOS.Dono
BARCOS.NomeBarco = CARGAS.Barco

CARGAS.Destino = PORTOS.NomePorto

CARGAS.Seguro = APOLICES.Apolice

APOLICES.Segurador = SEGURADORAS.NomeSegurador

As relações BARCOS e DONOS possuem um índice agrupado pelos atributos DONOS.NomeDono e BARCOS.Dono tornando bastante eficiente a localização dos barcos de um detrminado dono.

A base de conhecimento utilizada, expressa na forma de RIs, também é livre da especificação das junções, tal como as consultas:

R1 - (Capacidade > 350) -> (Facilidade = "offshore")

Todo Barco com capacidade de carga > 350 tonelaas pode operar somente nos portos que oferecem facilidade de carga e descarga "offshore".

R2 - (Capacidade > 300) -> (Negocio = "leasing")

Somente as companhias de "leasing" trabalham com navios com capacidade > 300.

R3 - (Cobertura <= Valor)

O valor do seguro de uma carga nunca pode ser maior que o seu valor em dinheiro.

R4 - (Quantidade <= Capacidade)

Um barco carrega no maximo uma carga igual a sua capacidade.

R5 - (TipoCarga E {"LNG","refinado"}) e (Valor > 500) ->

(Facilidade = "geral")

onde E = pertinência

Qualquer carga de "LNG" ou outro produto refinado que custe mais de 500 so pode ser tratada em portos que possuam facilidade "geral".

R6 - (Capacidade > 150) -> (TipoBarco = "supertanque")

Os Únicos barcos que possuem capacidade maior que 150 toneladas são os "supertanques".

R7 - (Valor > 3000) e (TipoBarco = "supertanque") ->

(Segurador = "Lloyd")

Cargas com valor acima de 3000 e transportadas por "supertanques" tem a sua apólice de seguro feitas pelo segurador "Lloyds".

R8 - (Negocio = "petroleo") ->

(TipoCarga E{"LNG", "refinado", "oleo"})

onde E = pertinência

Companhias trasportadoras cujo negócio é "petróleo" transportam somente "LNG", "refinado" e "oleo".

A consulta utilizada neste exemplo é:

(Capacidade > 400) e (TipoBarco = "supertanque") e
(Valor < 1000) e (Facilidade = "offshore"):(? Destino)

Neste exemplo, estão inicialmente envolvidas as relações BARCOS (Capacidade e TipoBarco), CARGAS (Valor e Destino) e PORTOS (Facilidade). Estas relações estão ligadas através das seguintes junções implicitas:

BARCOS.NomeBarco = CARGAS.Barco

CARGAS.Destino = PORTOS.NomePorto

Utilizando as heurísticas apresentadas sobre a consulta, as relações BARCOS, PORTOS e CARGAS são designadas alvo pela heurística H3 pois possuem junções com outras relações da consulta que ja estão suficientemente restringidas.

Pela heurística H5 estas relações continuam sendo alvo pois elas já fazem parte da consulta original não sendo necessária a sua introdução. Pela H6 elas também continuam sendo alvo pois não possuem um índice agrupado com as outras relações da consulta.

BARCOS e PORTOS são designados alvo também pela heurística H4, já que elas possuem junções somente com uma relação (CARGAS) e os seus atributos não contribuem para a resposta da consulta (Destino).

Pela heurística H2, a relação DONOS é designada como alvo, já que ela é bem menor que a relação BARCOS e possui um índice agrupado com esta.

As relações APOLICES e SEGURADORAS são designadas não-alvo pela heurística H5 pois não fazem parte da consulta original e não atenderam a H2.

Analisando as RIs, são selecionadas somente as regras R1 e R2, como sendo aplicáveis sobre as relações alvo

determinadas no passo anterior, produzindo estas novas restrições:

(Facilidade = "offshore"), pela restrição R1

(Negocio = "leasing"), pela restrição R2

As outras RIs não são aplicáveis por apresentarem restrições sobre as relações não alvo ou por envolverem variáveis não pertencentes a consulta.

Baseado nas restrições selecionadas, as modificações sobre a consulta original geram quatro alternativas, combinando a introdução e a eliminação de junções propostas pelas regras R1 e R2.

Podemos introduzir a junção com DONOS imposta pela regra R2 (Negocio = "leasing"), podemos eliminar a junção com PORTOS como determina a regra R1 (Facilidade = "offshore"), podemos fazer as duas modificações, ou podemos não alterar a consulta original.

As quatro consultas alternativas são geradas e a análise de custos é feita seguindo os metódos convencionáis de avaliação, aplicados nos sistemas de otimização de consultas de SGBDs relacionáis, determinando que ambas as modificações sejam realizadas, ficando assim a consulta:

(Capacidade > 400) e (TipoBarco = "supertanque") e
(Valor < 1000) e (Negocio = "leasing"):(? Destino)

Somente neste exemplo, houve um ganho real de mais de 20 vezes (de 439 para 19 segundos no tempo de execução), mostrando toda a potencialidade da otimização semântica sobre consultas a BD relacionais.

As informações sobre cardinalidade das relações e existência de índices, que são utilizadas no processo de otimização, são trazidas do catálogo do SGBD.

Contudo, é importante observar que o sistema QUIST trata apenas de RIs da categoria de restrições semânticas, não abordando restrições que expressem dependências convencionais.

3.2 - Otimização semântica de consultas em sistemas especialistas e em sistemas de banco de dados.

Em [CHAK86], é descrito um processo denominado Compilação Semântica, que consiste basicamente da incorporação da semântica expressa nas RIs às regras de dedução do BDI e na separação do processo de dedução de novos fatos do processo de acesso aos fatos armazenados no BDE. A aplicação do seu trabalho se estende a todos os BDDs que utilizem lógica de predicados de primeira ordem para representação das informações. Utilizando a definição de BDDs ele subdivide as informações ali armazenadas em três classes:

- Leis gerais. Conhecidos também como axiomas implícitos ou regras de dedução;

- Fatos ou axiomas explícitos;

- Restrições de integridade expressando a semântica.

No seu trabalho ele afirma que quando a base de fatos do BDD é pequena e existem poucas regras no BDI, a própria linguagem Prolog pode ser utilizada para a obtenção de respostas eficientemente. Contudo, quando o BDE se torna grande em relação ao BDI, é interessante separar o processo de inferência do processo de acesso aos dados, armazenando o BDE em um SGBD relacional.

O primeiro processo, denominado compilação, é um mecanismo de dedução que usa apenas os axiomas do BDI,

produzindo um novo conjunto de axiomas implícitos que fazem referência somente às relações armazenadas no BDE. O segundo processo faz, então, o acesso ao BDE, usando os métodos de acesso à bancos de dados relacionais.

No processo de compilação para cada predicado implícito do BDI é feita uma consulta ao mecanismo de dedução e todas as respostas possíveis para esta consulta são obtidas. Uma resposta é atingida se uma cláusula consistindo somente de predicados do BDE puder ser deduzida. Cada resposta corresponde a um novo axioma do BDI fazendo referência somente ao BDE. Note-se que não é necessário guardar todos os passos intermediários da dedução, apenas a cláusula original selecionada do BDI para compilação e as respostas finais geradas devem ser armazenadas. O novo axioma do BDI possui o antecedente do axioma original como consequente e os literais das respostas obtidas como antecedentes. Como exemplo, consideremos o seguinte BDD:

Esquema das relações do BDE (identificados por um *):

E1: ensi*(professor, departamento, curso)

E2: matr*(aluno, departamento, curso)

E3: dept*(departamento, curso, creditos)

Axiomas do BDI:

A1: Gilberto ensina em todos os cursos oferecidos pelo departamento de Química:

ensi(gilberto,X,Y) <-- dept*(X,Y,Z), X=quimica.

A2: Minker ensina em todos os cursos oferecidos pelo departamento de Informática.

ensi(minker,X,Y) <-- dept*(X,Y,Z), X=informatica.

A3: O Resto do axioma ensi é o mesmo que o armazenado no BDE na relação ensi*.

ensi(X,Y,Z) <-- ensi*(X,Y,Z).

A4: Se um professor X ministra um curso R pertencente ao departamento Y e um aluno Z esta matriculado no curso R no departamento Y então X é professor de Z.

professor_de(X,Z) <-- ensi(X,Y,R), matr*(Z,Y,R).

Para compilarmos o predicado do BDI "professor_de", seria formulada a consulta:

<-- professor_de(U1,U2).

Usando o mecanismo de dedução, pelo axioma A4, seria gerado o seguinte axioma intermediário:

ensi(U1,Y,R), matr*(U2,Y,R).

Aplicando novamente o processo de inferência sobre o predicado "ensi", seriam obtidos os seguintes axiomas:

dept*(Y,R,Z), matr*(U2,Y,R),U1=gilberto,Y=quimica

dept*(Y,R,Z),matr*(U2,Y,R),U1=minker,Y=informatica

```
ensi*(U1,Y,R), matr*(U2,Y,R)
```

Como todas os axiomas gerados estão formados exclusivamente por predicados do BDE, o processo de inferência termina dando origem aos novos axiomas compilados:

```
AC1: professor_de(U1,U2) <-- dept*(Y,R,Z),
                             matr*(U2,Y,R),
                             U1=gilberto,Y=quimica.

AC2: professor_de(U1,U2) <-- dept*(Y,R,Z),
                             matr*(U2,Y,R),
                             U1=minker,Y=informatica.

AC3: professor_de(U1,U2) <-- ensi*(U1,Y,R),
                             matr*(U2,Y,R).
```

Como pode ser visto, um axioma do BDI pode dar origem a vários novos axiomas no BDI compilado. Isto significa que o axioma pode ser obtido de diversas formas, correspondentes a cada cláusula de Horn gerada pelo mecanismo de dedução. A relação virtual correspondente ao predicado implícito é a união das relações produzidas para cada axioma que possua o predicado implícito como consequente.

Uma segunda parte deste trabalho descreve o algoritmo "Partial Subsumption", para incorporação da semântica expressa pelas RIs aos axiomas compilados do BDI, de tal forma que ela possa ser usada posteriormente para a otimização de consultas. O processo se baseia no algoritmo

"Subsumption", descrito em [CHAN73], para gerar "resíduos lógicos" quando uma RI é parcialmente assumida por um axioma compilado. Para ilustrar este processo vejamos um exemplo.

Todas as RIs utilizadas são implicações quantificadas universalmente, formadas por predicados do BDE e por predicados avaliáveis (=, >,...). Para este exemplo as RIs utilizadas são:

RI1: Braulio ensina somente cursos oferecidos pelo departamento de historia.

(X = historia) <-- ensi*(braulio, X, Y).

RI2: Todos os cursos do departamento de quimica são de 4 créditos.

(Z=4) <-- dept*(quimica, V, Z).

RI3: João esta matriculado em cursos do departamento de informatica ou de história.

(D=informatica), (D=historia) <-- matr*(joao,D,Y).

Por exemplo, o axioma AC3 e a restrição de integridade RI2 não têm predicados em comum, logo esta RI não é relevante para este axioma. Formalmente, é dito que RI2 não possui compatibilidade de união com o axioma AC3. Considerando agora o axioma AC3 e RI3, é possível incorporar a condição (D=informatica), (D=historia) <-- (U2=joao) ao axioma. Portanto, se durante uma consulta sobre o predicado "professor_de", a variável U2 for instanciada

para "joao", os consequentes (D=informatica) ou (D=história) poderão ser introduzidos na consulta e utilizados em um processo de otimização. O axioma AC3, com o resíduo incorporado, ficaria da seguinte forma:

```
ASR3: professor_de(U1,U2) <-- ensi*(U1,Y,R),
                              matr*(U2,Y,R),
          {(D=informatica), (D=historia) <-- (U2=joao)}.
```

A lista representada por {...} é chamada de lista de restrições e contém os resíduos gerados pelo algoritmo Partial Subsumption. A título de ilustração, são apresentados todos os axiomas semanticamente compilados, gerados a partir das RIs e dos axiomas compilados mostrados anteriormente:

```
ASR1: professor_de(U1,U2) <-- dept*(Y,R,Z),
                              matr*(U2,Y,R),
                              U1=gilberto,Y=quimica,
        {(Z=4)<--,  <-- (U2=joao)}.

ASR2: professor_de(U1,U2) <-- dept*(Y,R,Z),
                              matr*(U2,Y,R),
                              U1=minker,Y=informatica,
          {}.

ASR3: professor_de(U1,U2) <-- ensi*(U1,Y,R),
                              matr*(U2,Y,R),
          {(Y=hitoria) <-- (U1=braulio),
           (Y=informatica), (Y=hitoria) <-- (U2=joao)}.
```

As restrições de integridade também podem ser associadas às relações do BDE, para responder consultas que envolvam somente predicados do BDE. Associando as RI1, RI2 e RI3 a "matr*" e "dept*" obtemos:

SRE2: matr*(X,Y,Z)

{(Y=informatica),(Y=historia) <-- (X=joao)}.

SRE3: dept*(X,Y,Z)

{(Z=4) <-- (X=quimica)}.

Os algoritmos que compõem o processo de compilação semântica não serão apresentados, já que não foram utilizados no nosso trabalho. Uma descricação detalhada de todo o processo pode ser obtida em [CHAK86].

Este trabalho apresenta uma forma bastante interessante de integração das informações semânticas, representadas por RIs de ambas as categorias, aos axiomas de dedução do BDI. Contudo, não é apresentado nenhum esquema de utilização desses resíduos em um processo de otimização de consultas, que pudesse levar em conta os benefícios reais das modificações introduzidas na consulta original. A idéia apresentada na primeira parte do trabalho, onde foi descrito o processo de compilação do BDI, separando os mecanismos de inferência dos mecanismos de acesso aos dados, foi considerada bastante interessante e foi utilizada no nosso trabalho, como será descrito posteriormente.

3.3 - Uma interface otimizada entre Prolog e um sistema de consultas relacional.

Em [JARK84], é apresentado um mecanismo de tradução otimizada para a interação entre sistemas inteligentes baseados em lógica escritos em Prolog (Sintaxe de Edinburgh) e sistemas de bancos de dados relacionais, utilizando uma linguagem intermediária (DBCL).

Este mecanismo é composto de três processos básicos:

- Tradução de uma consulta Prolog em DBCL;
- Otimização semântica da consulta expressa em DBCL;
- Tradução da consulta otimizada expressa em DBCL em uma consulta em uma linguagem alvo (SQL).

A linguagem intermediária DBCL é formada por um sub-conjunto de Prolog, sem variáveis, para permitir instanciamento. As constantes da consulta original são mapeadas nelas mesmas, as variáveis universalmente quantificadas da cláusula meta original são traduzidas por um "t_", indicando que elas são atributos alvo ("target") da consulta, e as outras variáveis são precedidas por um "v_" e um número, distinguindo entre variáveis diferentes representando o mesmo atributo.

A DBCL para consultas conjuntivas é definida como:

dcbl(Esquema,Listalvo,Relreferências,Relcomparações).

"Esquema" é a lista dos atributos de todas as relações que compõem o esquema do banco de dados, além do nome do BD. Supondo, por exemplo, o BD "empdep" formado pelas relações empr e dept, teríamos o seguinte esquema:

empr(eno,nom,sal,dno).

dept(dno,fun,chf).

Esquema: [empdep,eno,nom,sal,dno,fun,chf].

"Listalvo" possui o mesmo formato de Esquema e define o esquema da relação resultante da chamada ao BD.

"Relreferências" é uma lista, onde cada elemento corresponde a uma linha com o mesmo formato do esquema, com "*" nos atributos não aplicáveis. Em termos de SQL, cada "relreferência" corresponde a uma variável de relação. Se símbolos correspondentes a variáveis (começando com v_ ou t_) aparecem várias vezes nas "relreferências", cada par corresponde a uma equi-junção.

"Relcomparações" é uma lista de listas, cada uma correspondendo a uma comparação relacional (maior, igual, ...). Cada sublista mapeia uma seleção ou uma teta-junção.

Para ilustrar o procedimento de conversão de Prolog para DBCL, vejamos um exemplo. Vamos supor a existência de uma cláusula Prolog "trabalha_dir_para", como apresentada abaixo:

```
trabalha_dir_para(X,Y) :-
        empr(_,X,_,D),
        dept(D,_,M),
        empr(M,Y,_,_).
```

Os "underscores" representam variáveis distintas, mas irrelevantes.

Suponhamos, agora, a existência da seguinte consulta: "quem trabalha diretamente para Sueli e ganha menos de 40000 ?":

```
:- trabalha_dir_para(X,sueli),
   empr(_,X,S,_),
   less(S,40000).
```

A consulta expressa em DBCL gerada seria:

```
dbcl(
/* Esquema
[empdep,    eno,    nom,    sal,    dno,    fun,    chf],

/* Listalvo
[trabalha_dir_para,
             *,    t_X,      *,      *,      *,      *],

/* Relreferências
[[empr,  v_Eno1,    t_X, v_Sal1,    v_D,      *,      *],
 [dept,       *,      *,      *,    v_D, v_fun2,    v_M],
 [empr,     v_M,  sueli, v_Sal3, v_Dno3,      *,      *],
 [empr,  v_Eno4,    t_X,    v_S, v_Dno4,      *,      *]],
```

```
/* Relcomparações
[[less,v_S,40000]]).
```

Neste trabalho, a função principal do meta-avaliador é a simulação do processo de dedução do Prolog, para traduzir as consultas feitas sobre os axiomas de dedução do Prolog em DBCL. Vejamos mais um exemplo que ilustra bem esta afirmação.

Considere esta outra cláusula Prolog e a consulta subsequente:

```
mesmo_chefe(X,Y) :-
        trabalha_dir_para(X,M),
        trabalha_dir_para(Y,M),
        neq(X,Y).

:- mesmo_chefe(X,joao).
```

Esta consulta em DBCL ficaria:

```
dbcl(
/* Esquema
[empdep,    eno,    nom,    sal,    dno,    fun,    chf],

/* Listalvo
[mesmo_chefe,
                *,    t_X,      *,      *,      *,      *],

/* Relreferências
[[empr,  v_Eno1,    t_X, v_Sal1,   v_D1,      *,      *],
```

```
[dept,      *,      *,      *,   v_D1, v_fun2,   v_M1],
[empr,   v_M1,    v_M, v_Sal3, v_Dno3,      *,      *],
[empr, v_Eno4,   joao, v_Sal4,   v_D2,      *,      *],
[dept,      *,      *,      *,   v_D2, v_fun5,   v_M2],
[empr,   v_M2,    v_M, v_Sal6, v_Dno6,      *,      *]],

/* Relcomparações
[[neq,t_X,joao]]).
```

O segundo processo de tradução consiste na conversão da consulta em DBCL em uma linguagem alvo (SQL). Como são tratadas somente consultas conjuntivas livres de funções, as consultas geradas não possuem encadeamento. As informações são extraídas da consulta em DBCL e utilizadas na montagem do comando, de acordo com as seguintes regras:

- Cada linha de Relreferências corresponde à definição de uma variável na cláusula FROM da consulta;

- Entradas que compõem a Listalvo aparecem na cláusula SELECT junto com o nome da variável apropriada (número da primeira linha em Relreferências onde aparece a entrada);

- As constantes de Relreferências são traduzidas em seleções na cláusula WHERE, com operador de comparação "=" e com o lado esquerdo formado pela linha (nome variável) e a coluna (nome do atributo) de onde a constante for localizada;

- Cada par de símbolos iguais em Relreferências,

começando com "v_" ou "t_" é traduzido em uma equi-junção da cláusula WHERE, os componentes são determinados por sua localização, como no passo anterior;

- Cada linha em Relcomparações é mapeada em uma seleção ou junção da cláusula WHERE. Os nomes das variáveis participantes e dos atributos são determinados pela localização da primeira ocorrência dos mesmos simbolos em Relreferências;

- Variáveis ou símbolos não repetidos não aparecem na consulta SQL.

Seguindo as regras apresentadas, a consulta em DBCL do exemplo anterior seria traduzida para:

```
SELECT v1.nom
FROM empr v1, dept v2, empr v3,
     empr v4, dept v5, empr v6
WHERE (v1.dno = v2.dno) AND (v2.chf = v3.eno) AND
      (v4.dno = v5.dno) AND (v5.chf = v6.eno) AND
      (v4.nom = "joao") AND (v3.nom = v6.nom) AND
      (not v1.nom = "joao");
```

O processo de otimização não é feito por uma simples reordenação dos predicados nas cláusulas Prolog, já que isto continua sendo feito no processo de otimização convencional pelo SGBD (reordenação das operações relacionais). Aqui são utilizadas informações semânticas expressas em RIs da

categoria de dependências convencionais, para realizar modificações na consulta em DBCL, visando obter uma melhora de desempenho na execução da consulta.

Para cada classe de RI, é apresentado um procedimento especifico. São três as classes tratadas: limites para valores ("value bounds"), dependências funcionais e integridades referenciais.

Na primeira classe, os limites para valores são introduzidos em Relcomparações, visando detectar contradições ou comparações redundantes. Eles são definidos através de predicados "valuebound". Para o nosso BD exemplo, teríamos o predicado:

```
valuebound(empr, sal, 10000, 90000),
```

significando que o salário de todos os empregados estão no intervalo de 10000 a 90000.

Na segunda classe, as dependências funcionais são utilizadas na renomeação de variáveis na tentativa de igualar linhas em Relreferências, tornando uma delas removível. As dependências são especificadas em termos de predicados "funcdep". Para o nosso BD exemplo, teríamos os seguintes predicados:

```
funcdep(empr, [nom], [eno]).
funcdep(empr, [eno], [nom,sal,dno]).
funcdep(dept, [dno], [fun,chf]).
```

funcdep(dept, [chf], [dno]).

O primeiro predicado indica que nome do empregado determina funcionalmente o número do mesmo na relação empr. O segundo e o terceiro, determinam que número do empregado e número do departamento são chaves das relações empr e dept, respectivamente. O quarto determina que o número do chefe do departamento determina funcionalmente o número do departamento.

Na terceira classe das RIs utilizadas, estão as integridades referenciais, que são utilizadas para eliminar linhas removiveis de acordo com o seguinte critério: caso exista uma linha r em Relreferências, com um conjunto R de atributos que pode ser dividido em dois grupos R1 e R2, sendo que os atributos em R1 não aparecem em outro lugar, nem são atributos alvo (começam com "v_"), e existe uma outra linha r´ com um conjunto de atributos R´, subdividido em R1´ e R2´, de tal forma que R2 = R2´, a linha r é removível desde que exista um predicado "refint" com as seguintes características:

refint(r´,[R1´],r,[R1]).

Para o nosso BD exemplo, teríamos os seguintes predicados:

refint(empr, [dno], dept, [dno]).
refint(dept, [chf], empr, [eno]).

significando que todo número de departamento na relação empr

deve corresponder a um número de departamento na relação dept, e que todo chefe na relação dept deve corresponder a um número de empregado na relação empr.

O algoritmo completo de otimização pode ser resumido nos seguintes passos:

Início;

Adicione os limites para valores ao conteúdo de Relcomparações, para variáveis de atributos que lá apareçam, e verifique se todas as constantes aparecendo em Relreferências não apresentam contradições com o seu seu domínio. Caso sejam encontradas contradições, encerre a consulta com uma resposta vazia;

Repita

Aplique a simplificação das desigualdades. Se uma contradição for detectada, pare a consulta com uma resposta vazia;

Aplique as dependências funcionais, renomeando as variáveis e removendo as linhas que ficarem duplicadas. Se uma contradição for detectada, pare a consulta com uma resposta vazia;

até que não tenha havido renomeação de variáveis;

Aplique as integridades referenciais, eliminado as linhas removíveis;

Fim.

Aplicando este processo de otimização sobre a consulta exemplo em DBCL, utilizando as RIs apresentadas, veremos inicialmente que não existem limites para valores aplicáveis. Contudo, as dependências funcionais levam a eliminação de duas linhas em Relreferências. Pela primeira dependência apresentada, as variáveis v_M1 e V_M2, respectivamente na terceira e sexta linhas, podem ser igualadas, já que ambas possuem a variável v_M na coluna "nom", que determina funcionalmente a coluna "eno".

A coluna eno é uma chave da relação empr, como foi apresentado na segunda dependência funcional, portanto, as duas linhas descrevem o mesmo conjunto de empregados, podendo ser eliminada a sexta linha em Relreferências. Neste processo, o atributo chf na quinta linha foi renomeado para v_M1 e, como ele determina funcionalmente a coluna "dno" (quarta dependência apresentada), as variáveis v_D1 e v_D2 podem ser igualadas.

Usando, então, a terceira dependência, as duas linhas podem ser igualadas, o que acarreta a eliminação da quinta linha. Neste ponto, é encerrado o loop principal do algoritmo apresentado e a consulta atual em DBCL pode ser vista abaixo:

```
dbcl(
[empdep,    eno,    nom,    sal,    dno,    fun,    chf],

[mesmo_chefe,
```

```
                *,     t_X,      *,      *,      *,      *],

[[empr,  v_Eno1,    t_X, v_Sal1,   v_D1,      *,      *],
 [dept,       *,      *,      *,   v_D1, v_fun2,   v_M1],
 [empr,    v_M1,    v_M, v_Sal3, v_Dno3,      *,      *],
 [empr,  v_Eno4,   joao, v_Sal4,   v_D1,      *,      *]],

[[neq,t_X,joao]]).
```

Aplicando, agora, as integridades referenciais, veremos que a terceira linha é removível, pois a variável v_M1 aparece na segunda linha, e as outras variaveis v_M, v_Sal3 e v_Dno3 não aparecem em outro lugar em Relreferências, e existe um predicado refint diretamente aplicável entre o atributo chf da relação dept e o atributo eno na relação empr. Após a eliminação da segunda linha, a terceira linha torna-se removível por motivos semelhantes aos anteriores. A consulta final em DBCL gerada no processo de otimização pode ser vista abaixo:

```
dbcl(
[empdep,    eno,    nom,    sal,    dno,    fun,    chf],

[mesmo_chefe,
                *,     t_X,      *,      *,      *,      *],

[[empr,  v_Eno1,    t_X, v_Sal1,   v_D1,      *,      *],
 [empr,  v_Eno4,   joao, v_Sal4,   v_D1,      *,      *]],

[[neq,t_X,joao]]).
```

Analisando a consulta inicial e final em DBCL, nós concluímos que a utilização das informações semânticas expressas nas RIs permitiu alterar a consulta "Quem trabalha para o mesmo chefe do João" para "Quem trabalha no mesmo departamento de João". Realizando a conversão da DBCL final para SQL, podemos notar que das cinco junções que faziam parte da consulta original, restou apenas uma na consulta otimizada:

```
SELECT v1.nom
FROM empr v1, empr v2
WHERE (v1.dno = v2.dno) AND (v2.nom = "joao")
                        AND (not v1.nom = "joao");
```

Deste trabalho, foram utilizadas algumas idéias relacionadas com o processo de otimização e com o processo de conversão da consulta em DBCL para SQL, como será visto posteriormente. Contudo, deve-se notar que foram tratadas apenas RIs da categoria das dependências convencionais, não sendo abordadas RIs da categoria das restrições semânticas.

IV - Apresentação do Otimizador Semântico

Nossa pesquisa foi desenvolvida com o intuito de obter um otimizador semântico de consultas que utilizasse as informações semânticas expressas nas RIs, realizando uma análise baseada em heurísticas, para determinar que modificações propostas trariam realmente alguma melhora de desempenho para a consulta.

O otimizador semântico atua sobre consultas formuladas a um banco de dados dedutivo, expresso em Prolog, e é acoplado ao processo de tradução da consulta Prolog para uma linguagem de acesso a banco de dados (SQL).

A opção pela utilização de bancos de dados dedutivos surgiu pelas próprias características de tais sistemas, uma vez que eles são modelados de tal forma que os fatos (axiomas explícitos do BDE) e as regras de dedução (axiomas implícitos do BDI) podem ser tratados de forma distinta. Os dados são armazenados em um gerenciador externo e recuperados para responder às consultas do usuário, e os axiomas de dedução são tratados por um processo de "compilação" do BDI semelhante ao apresentado em [CHAK86], separando o processo de inferência do processo de acesso aos dados possibilitando a introdução do mecanismo de otimização entre os dois processos.

A linguagem Prolog foi escolhida para representar os BDDs, seguindo uma tendência dos principais trabalhos nesta área e também pela existência de uma formalização no emprego de Prolog na representação de BDDs [LLOY85].

Uma outra característica necessária para o otimizador foi que ele fosse capaz de tratar as duas categorias de restrições de integridade apresentadas, restrições semânticas e dependências convencionais. Isto foi conseguido utilizando uma representação padrão para todas as RIs (predicados em lógica de primeira ordem sob a forma de cláusulas de Horn), expressas também em Prolog, como foi mostrado nas seções 2.1 e 2.4.

As heurísticas utilizadas no processo de otimização semântica foram extraidas do trabalho de King [KING81] e expandidas para atender às duas categorias de RIs abordadas. Elas expressam meta-informações sobre o banco de dados dedutivo e guiam todo o processo de modificação da consulta original.

A estrutura geral do sistema é apresentada abaixo:

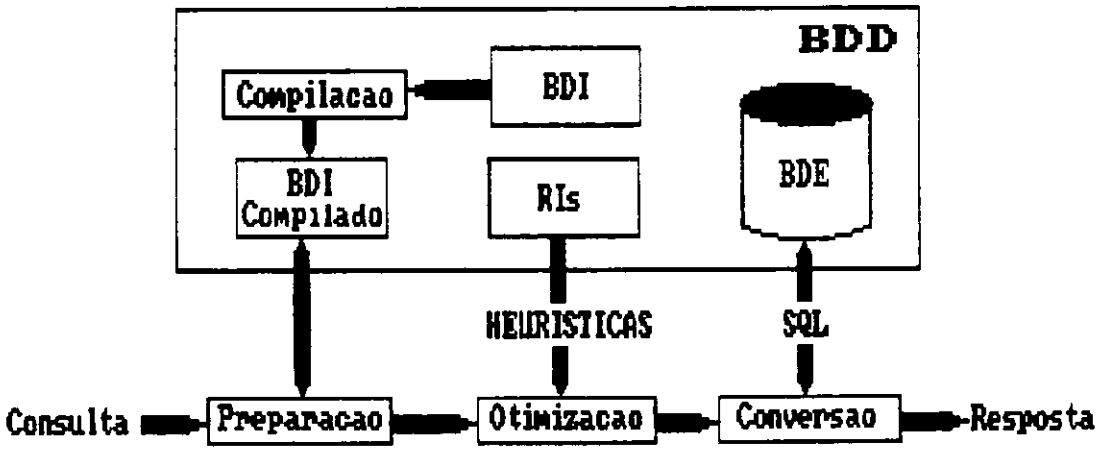


Fig. 1 - Estrutura Geral do Otimizador

A seguir, são descritos todos os processos que compõem o sistema apresentado na Fig. 1.

4.1 - Compilação do BDI

O processo de compilação do BDI consiste na transformação de todas as regras de dedução do BDI, de tal forma que elas fiquem compostas, apenas, por predicados do esquema do BDE e por predicados de comparação. Este processo pode ser visto como uma meta-avaliação do BDI, já que ele simula o processo de inferência do Prolog, como foi visto no trabalho de Chakravarthy apresentado na seção 3.2, e consiste basicamente das seguintes etapas:

- Geração de uma consulta para cada conseqüente distinto dos axiomas do BDI;

- Obtenção da árvore de prova para as consultas geradas no passo anterior. A árvore de prova é obtida com a aplicação sucessiva do mecanismo de inferência sobre as consultas, até que seja atingido um nó folha contendo somente referências aos predicados do BDE;

- O conseqüente dos novos axiomas é formado pelo nó topo de cada árvore gerada (conseqüente do axioma que deu origem à consulta) e os antecedentes são obtidos dos nós folhas da árvore de prova.

Desta descrição, pode-se notar que a compilação de um axioma do BDI pode gerar vários novos axiomas, cada um correspondendo a uma solução encontrada para a consulta. Pode-se notar também que esta abordagem para o problema da compilação não trata axiomas de dedução com definição

recursiva.

Como exemplo, a compilação do axioma de dedução "mesmo_gerente" do BDD apresentado na seção 2.2 resulta em:

```
mesmo_gerente(A,B) :-
        empE(C,A,D,E),
        deptE(E,F,G),
        empE(G,H,I,J),
        empE(K,B,L,M),
        deptE(M,N,O),
        empE(O,H,P,Q),
        A \= B.
```

Note-se que o axioma mesmo_gerente ficou composto somente por referências aos predicados do BDE e por desigualdades.

Este procedimento de compilação é ativado somente quando ocorre alguma modificação na estrutura do BDD, já que as consultas serão feitas sobre o BDI já compilado.

4.2 - Preparação da consulta

Este processo cuida da recepção e preparação da consulta para a otimização. Está subdividido em três etapas:

- Recebe uma consulta Prolog, que pode fazer referências aos predicados do esquema do BDE e/ou aos axiomas de dedução do BDI;

- Monta um axioma "resposta", a partir da consulta do usuário. Esta operação determina quais são as variáveis cujos valores o usuário deseja obter como resposta à sua consulta. Estas variáveis são identificadas na consulta por letras maiúsculas previamente estabelecidas (X, Y, Z,...);

- Compila o axioma resposta gerado no passo anterior, utilizando o BDI previamente compilado. Este processo consiste apenas em uma avaliação parcial das referências aos predicados do BDI que existam na consulta. Esta avaliação é semelhante à usada no processo de compilação do BDI, contudo o processo de inferência necessita apenas de um passo para encontrar a solução, já que o BDI foi compilado anteriormente.

Vejamos como ficaria o axioma resposta, montado para a consulta "Quais os nomes dos empregados que possuem o mesmo chefe de João e que recebem salário maior que 500":

```
:- mesmo_gerente(X,joao),
   empregado(_,X,S,_),
   S > 500.

   resposta(X) :-
        empE(C,X,D,E),
        deptE(E,F,G),
        empE(G,H,I,J),
```

```
empE(K,joao,L,M),
deptE(M,N,O),
empE(O,H,P,Q),
X \= joao,
empE(R,X,S,T),
S > 500.
```

Vale notar que, para este exemplo, seriam gerados tantos predicados resposta quantos fossem os predicados na definição do axioma mesmo_gerente.

4.3 - Otimização da Consulta

Este processo recebe uma consulta compilada e realiza a otimização semântica, gerando uma nova consulta. A otimização segue os seguintes passos:

- Converte a consulta (expressa nos axiomas resposta) para o mesmo formato das RIs (lógica de primeira ordem) com os antecedentes formados pelo corpo da consulta e o conseqüente pelo axioma resposta.

- Verifica quais RIs são aplicáveis. Esta verificação é feita provando-se os antecedentes de uma RI, com base na consulta. Este provador é facilmente implementado, como veremos no capítulo 5, pois as RIs são conjunções formadas apenas por predicados de referência ao BDE e predicados de comparação;

- Analisa as modificações determinadas pelos

conseqüentes das RIs aplicáveis, de acordo com heurísticas, visando determinar suas vantagens reais na melhora do desempenho da consulta;

- Realiza as modificações selecionadas no passo anterior. Cada classe de RIs determina um tipo de modificação diferente de acordo com uma heurística especifica;

- Caso ainda existam RIs aplicáveis na nova consulta, volta para o segundo passo.

4.3.1 - Heurísticas Utilizadas

No processo de otimização, algumas heurísticas são utilizadas para determinar quais RIs devem ser aplicadas na modificação da consulta.

As heurísticas utilizadas são:

H1 - Substituição de Variáveis: "Aplique-se uma RI cujo conseqüente seja formado por igualdades". Esta heurística visa contemplar as dependências funcionais e as dependências de chave. Sua aplicação iguala variáveis da consulta, podendo tornar alguns predicados idênticos, permitindo a eliminação de um deles. A melhora no desempenho é trazida pela eliminação da junção correspondente. Como exemplo vejamos a seguinte consulta realizada sobre o BDD exemplo apresentado em 2.4:

```
:- empE(Matr,Nome1,Sal1,_),
   empE(Matr,Nome2,Sal2,_),
   Sal1 \= Sal2.
```

Quais os empregados de mesma matricula que possuem salários diferentes ?. Aplicando a RI3 (dependência de chave) as variáveis Nome1 e Nome2 são igualadas bem como as variáveis Sal1 e Sal2, modificando a consulta para:

```
:- empE(Matr,Nome1,Sal1,_),
   empE(Matr,Nome1,Sal1,_),
   Sal1 \= Sal1.
```

Como os dois predicados "empregado" são idênticos um deles pode ser eliminado reduzindo a consulta a:

```
:- empE(Matr,Nome1,Sal1,_),
   Sal1 \= Sal1.
```

Onde pode-se notar uma contradição e portanto a consulta tem resposta vazia.

H2 - Eliminação de Restrições: "Aplique-se uma RI que determine a eliminação, na consulta, de uma restrição que possa ser inferida de uma desigualdade no conseqüente da RI". Esta heurística contempla os limites para valores e as informações genéricas que determinem desigualdades. A eliminação da seleção redundante produz uma melhora no desempenho da consulta. Como exemplo vejamos a seguinte

consulta:

```
:- empE(Matr,Nome,Sal,_),
   Sal < 12000.
```

Quais os empregados com salário menor que 12.000 ?. Aplicando a RI7 (limites para valores) que determina que o salário de todos os empregados são menores que 10.000 a restrição da consulta pode ser removida pois é redundante, reduzindo a consulta a:

```
:- empE(Matr,Nome,Sal,_).
```

H3 - Detecção de Contradição: "Aplique-se uma RI cujo conseqüente seja uma desigualdade que determine uma contradição em relação às restrições da consulta". Neste caso, o processo de otimização é encerrado e uma mensagem de resposta nula é apresentada ao usuário. Como exemplo vejamos a consulta:

```
:- empE(Matr,Nome,Sal,_),
   Sal > 12.000.
```

Quais os empregados com salário maior que 12.000 ?. Aplicando a RI7 (limites para valores) que determina que o salário de todos os empregados são menores que 10.000, é encontrada uma contradição com a seleção na consulta e portanto a resposta a consulta é vazia.

H4 - Introdução de Restrição: "Aplique-se uma RI que determine a inclusão de uma restrição em uma variável da consulta, desde que exista um índice por esta variável no BDE". Uma melhora no desempenho é conseguida já que a pesquisa por um índice é melhor que uma pesquisa seqüencial sobre uma relação do BDE. Como exemplo vejamos a consulta:

:- empE(Matr,Nome,Sal,4).

Quais os empregados do departamento de número 4 ?. Aplicando a RI9 (restrição semântica) determinando que todo empregado do departamento 4 ganha mais de 8.000 e considerando a existência de um índice por salário na relação empE do BDE a seleção da RI seria incluída na consulta que ficaria:

:- empE(Matr,Nome,Sal,4),
   Sal > 8000.

H5 - Eliminação de Junções: "Aplique-se uma RI cujo conseqüente seja um predicado que possa ser eliminado da consulta". Esta heurística contempla as integridades referenciais e informações sobre o BDE. A melhora de desempenho é introduzida pela eliminação da junção redundante correspondente. A verificação para se saber se um predicado é redundante foi descrita anteriormente na

seção 3.3 e pode ser vista com mais detalhes em [JARK84] e [SHEN87]. Como exemplo vejamos a consulta:

```
:- empE(Matr,carlos,Sal,Dep),
   deptE(Dep,_,_).
```

Quais os empregados de nome "carlos" que estão alocados a departamentos existentes. Aplicando a RI5 (integridade referencial) que garante que todos os empregados estão alocados a departamentos existentes, a junção com deptE pode ser eliminada, reduzindo a consulta a:

```
:- empE(Matr,carlos,Sal,Dep).
```

H6 - Introdução de Junções: "Aplique-se uma RI cujo conseqüente determine a introdução de um predicado de junção, desde que as relações envolvidas estejam armazenadas juntas fisicamente e que exista um índice agrupado ("cluster index") pelo atributo de junção". A existência de índices agrupados é muito importante para o desempenho de uma consulta, pois significa que a seqüência lógica do índice equivale à seqüência física da relação. Ainda assim, uma avaliação da cardinalidade das relações envolvidas é necessária. Como exemplo vejamos a consulta:

```
:- empE(Matr,Nome,Sal,Dep),
```

```
   Sal > 8.000.
```

Quais os empregados que possuem salário maior que 8.000 ?. Aplicando a RI8 (restrição semântica) que determina que todo empregado que ganha mais de 8000 trabalha na presidência e admitindo a existência de um índice agrupado entre os números de departamento em empE e deptE, a junção com deptE pode ser incluída na consulta que ficaria:

```
:- empE(Matr,Nome,Sal,Dep),
   deptE(Dep,presidencia,_),
   Sal > 8.000.
```

A heurística H1 foi criada para garantir a aplicação das dependências funcionais e as dependências de chave tratadas no nosso trabalho. As heurísticas H2 e H3 foram criadas com base nas pesquisas apresentadas em [JARK84] e [JARK86]. As heurísticas H4, H5 e H6 foram extraídas e adaptadas do trabalho apresentado em [KING81].

Como exemplo, vejamos a aplicação do processo de otimização sobre o predicado resposta, montado no passo anterior, utilizando as RIs apresentadas na seção 2.2. Inicialmente, é selecionada a RI1 (dependência funcional) pela heurística H1 (Substituição de variáveis), para ser aplicada sobre a variável "X" na primeira e na oitava linhas do axioma resposta, igualando as variáveis "R" e "C".

A mesma RI é aplicada sobre a variável "H" na terceira e sexta linhas igualando as variáveis "G" e "O".

Em seguida a RI3 (dependência de chave) se torna aplicável pela mesma heurística H1, igualando as variáveis "D" e "E" na primeira linha às variáveis "S" e "T" na oitava linha. Com isto, a oitava linha pode ser removida pois tornou-se igual à primeira. Aplicando este mesmo processo sobre a terceira e sexta linha elas se tornam iguais, permitindo a eliminação da sexta linha. Neste ponto, o predicado resposta está assim:

```
resposta(X) :-
        empE(C,X,D,E),
        deptE(E,F,G),
        empE(G,H,I,J),
        empE(K,joao,L,M),
        deptE(M,N,G),
        X \= joao,
        D > 500.
```

Prosseguindo, teremos que a RI2 (dependência funcional) torna-se aplicável sobre a segunda e quinta linha igualando as variáveis "E" e "M". Em seguida, a RI4 (dependência de chave) é aplicável sobre estas mesmas linhas igualando as variáveis "N" e "F", tornando as duas linhas iguais e permitindo a eliminação da quinta linha.

Em seguida, a RI6 (integridade referencial) torna-se

aplicável pela heurística H5 (eliminação de junções) sobre a segunda e a terceira linhas, permitindo a eliminação da terceira. Da mesma forma, torna-se aplicável a RI5 (integridade referencial), baseando-se na mesma heurística, sobre a primeira e a segunda linhas, permitindo a eliminação da segunda linha da consulta. Agora, o predicado resposta está como abaixo:

```
resposta(X) :-
        empE(C,X,D,E),
        empE(K,joao,L,M),
        X \= joao,
        D > 500.
```

Neste ponto da otimização, apenas a RI7 (limites para valores) é aplicável pela heurística H2 (eliminação de restrição) permitindo a eliminação da seleção "D > 500", já que pela RI7 todos os salários são sempre maiores que 1000.

Como não existem mais RIs aplicáveis, o processo de otimização termina e o predicado resposta final obtido fica:

```
resposta(X) :-
        empE(C,X,D,E),
        empE(K,joao,L,M),
        X \= joao.
```

3.4 - Conversão e Execução da Consulta

Este processo é responsável pela conversão da consulta já otimizada, para uma consulta correspondente em SQL [DATE85], para ser finalmente executada. Passos seguidos:

- Recebe uma consulta otimizada e a converte para SQL, seguindo as idéias propostas por Jarke, apresentadas na seção 3.3 e utilizadas também em [CHAN86] e [GHOS88].

- Submete a consulta gerada ao BDE (SGBD Relacional);

- Recebe os resultados e os apresenta ao usuário.

Seguindo com o nosso exemplo, temos que a consulta SQL, gerada para o predicado resposta otimizado, fica como abaixo:

```
select v2.nome
from empE v1, empE v2
where (v1.dep = v2.dep) and
      (not v2.nome = 'joao') and
      (v1.nome = 'joao');
```

Como podemos ver, o processo de otimização reduziu as seis junções existentes na consulta original para apenas uma e ainda eliminou uma seleção desnecessária, trazendo um ganho bastante grande para o desempenho da consulta, como poderá ser visto no capítulo VI.

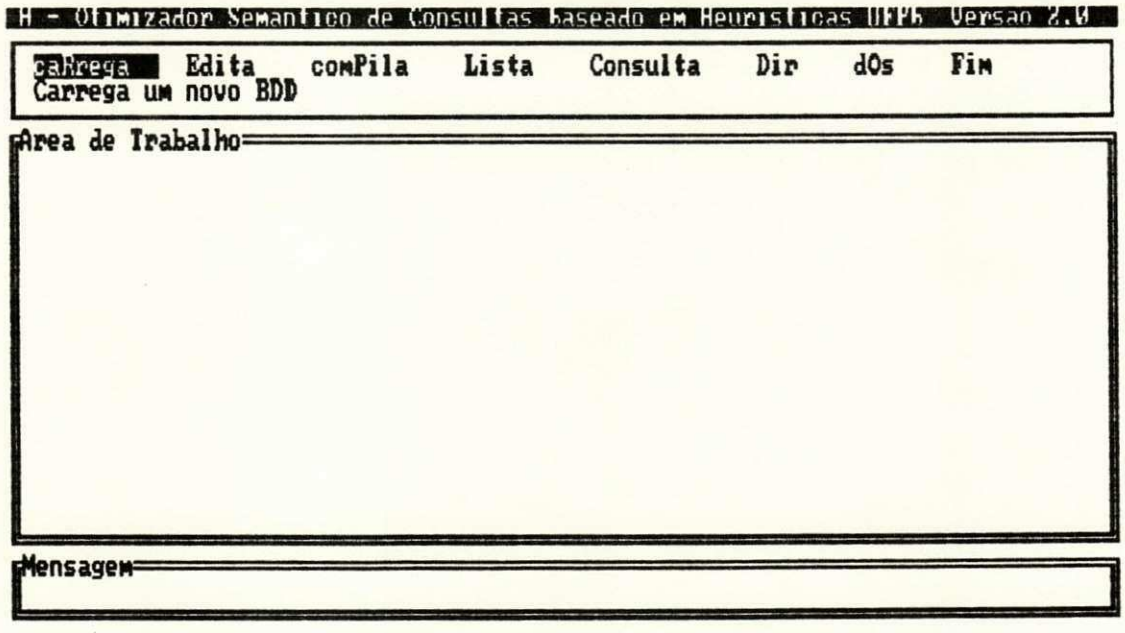
V - Implementação do Otimizador Semântico

Neste capítulo, é apresentada, inicialmente, a implementação da interface utilizada pelo sistema e, a seguir, serão descritas as implementações das operações disponíveis na interface.

O projeto apresentado encontra-se implementado em Prolog (sintaxe de Edinburgh), utilizando um micro da linha PC/XT, com 640K de memória principal. O BDE foi implementado no SGBD relacional "Oracle" [ORAC84].

5.1 - A interface

A interface do sistema com os usuários é feita através de uma única tela que está subdividida em três janelas, como pode ser visto abaixo:



As janelas foram criadas usando facilidades disponíveis na versão da linguagem Prolog utilizada ("gera_janela"). O

nome do axioma que implementa cada operação é apresentado entre parênteses, ao final de cada descrição. Cada janela possui funções bem distintas:

- Menu. Permite ao usuário selecionar uma operação que será executada pelo sistema. Sua implementação se utiliza de um menu horizontal circular (padrão 123), sobre o qual o usuário pode deslocar o cursor e selecionar uma opção pelo posicionamento do cursor ou pela letra maiúscula que identifica cada opção ("mcmenu").

- Area de Trabalho. Nesta janela, são tratadas as principais interfaces com o usuário, tais como: receber informações do usuário sobre o BDD a ser utilizado, receber consultas para otimização, apresentar para o usuário as consultas otimizadas e as respostas obtidas, apresentar listagens do BDD atual e apresentar o diretório dos BDDs existentes.

- Mensagens. Aqui são apresentadas as mensagens que o sistema envia para o usuário. Mensagens que especificam condições de erro são apresentadas junto com um sinal sonoro ("erro"), e a execução do sistema é retomada de um ponto previamente estabelecido ("inicio"). As mensagens de aviso são simplesmente escritas e removidas ("escreve_mensagem" e "limpa_mensagem").

5.2 - Carga do BDD

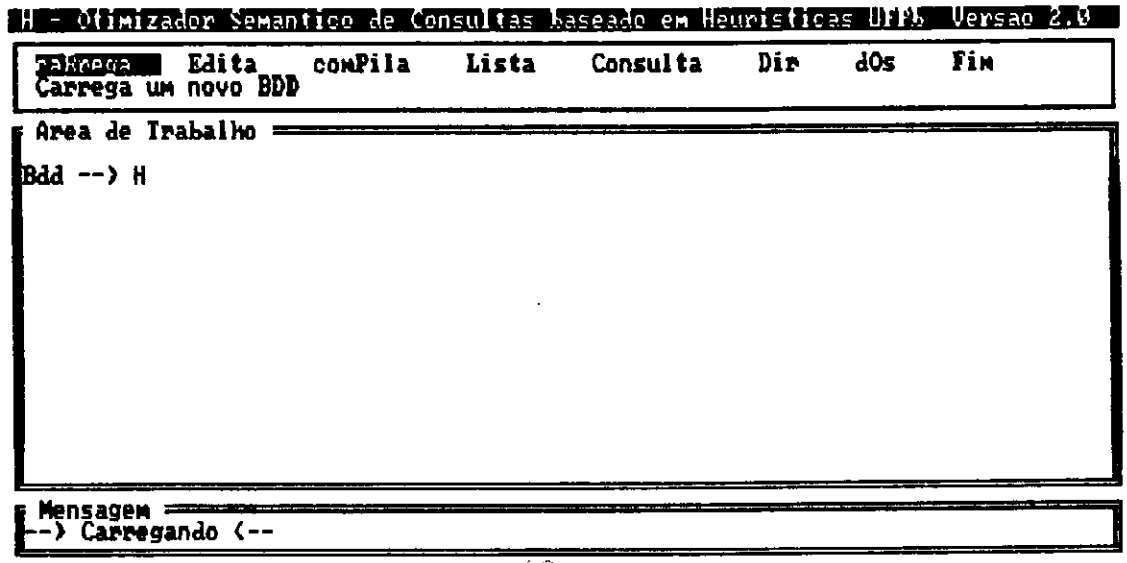
Este procedimento do sistema cuida da carga de um BDD

selecionado pelo usuário ("carrega_bdd"), é composto das seguintes operações:

- Recebe do usuário o nome do BDD com o qual ele deseja trabalhar ("read_string");
- Verifica a existência do BDD selecionado no diretório corrente ("directory");
- Cria uma área de código especifica para os BDDs ("create_world");
- Carrega o BDD selecionado na área de código criada ("reconsult").

Foram utilizadas áreas de código distintas para o código que compõem o otimizador e para os axiomas do BDD, visando permitir uma rápida recuperação dos predicados do BDD nos processos de compilação do BDI e listagem do BDD.

Como exemplo, podemos ver a carga do BDD "H" pelo sistema:



5.3 - Edição do BDD

Este procedimento permite que o usuário altere o BDD corrente utilizando um editor externo, previamente especificado por um comando "set editor", colocado no arquivo "autoexec" do disco corrente. O BDD alterado é recarregado na área de código apropriada, quando termina a edição ("edit").

5.4 - Compilação do BDI

O processo de compilação do BDI ("compila"), apresentado anteriormente, foi implementado utilizando idéias sugeridas por Minker, em [MINK88]. Os passos que compõem este processo são:

- leitura do BDD para identificar os axiomas do BDI que serão compilados. Uma lista é montada com todos os axiomas de dedução do BDI ("le_bdd").

- obtenção do nível de cada axioma de dedução da lista montada no passo anterior. Os predicados que compõem o esquema do BDE são de nível 1. O nível de uma regra de dedução é determinado pelo maior nível dos predicados que a compõem, acrescido de 1. E produzida uma lista, onde cada elemento corresponde a um axioma de dedução e o seu respectivo nível ("obtem_nivel").

- classifica-se a lista ("sort"), produzida no passo

anterior, pelos niveis e, para cada axioma da lista:

. obtém as definições do axioma, removendo-as temporariamente ("obtem_definicoes_basicas");

. substituem-se, nas definições obtidas no passo anterior, os predicados que possuem nivel maior que 2, por sua definição (que já estará compilada). Como um predicado pode possuir várias definições, podem ser gerados vários axiomas compilados para o mesmo axioma de dedução ("traduz");

. Grava os novos axiomas formados a partir das substituições ("grave_nova_definicao").

Ao final deste processo, todas as regras de dedução do BDI estarão definidas somente sobre predicados de nivel 1 (esquema do BDE) e sobre predicados de comparação (>, <, =, ...).

Estes passos descrevem uma forma de implementação bastante eficiente para as idéias descritas em [CHAK86], apresentadas anteriormente na seção 3.2. O processo de obtenção dos níveis é recursivo, já que para a obtenção do nível de um axioma é necessário se conhecer os níveis dos predicados que o compõem. O processo de geração dos novos axiomas é iterativo, gerando para cada axioma do BDI, um conjunto de novos axiomas por substituição de definições.

Abandonando um pouco o nosso BDD exemplo, vejamos como

seria a processo de compilação para os axiomas abaixo:

```
avos(X,Y) :- pais(X,Z), pais(Z,Y).
pais(X,Y) :- paiE(X,Y).
pais(X,Y) :- maeE(X,Y).
```

Supondo que os predicados avos e pais compõem o BDI e que os predicados paiE e maeE compõem o BDE ("E" no final do nome), nós teríamos que os predicados pai e mae são de nivel 1, o axioma pais é de nível 2, já que é composto somente por predicados de nivel 1, e o axioma avos é de nivel 3, já que ele é composto por predicados de nível 2.

Como os predicados pai, mae e pais já estão compostos somente por predicados do BDE (níveis menores ou igual a 2), não necessitam de substituição. Para o axioma avos são geradas as combinações possiveis por substituição do predicado pais por suas definições, gerando os novos axiomas:

```
avos(X,Y) :- paiE(X,Z), paiE(Z,Y).
avos(X,Y) :- paiE(X,Z), maeE(Z,Y).
avos(X,Y) :- maeE(X,Z), paiE(Z,Y).
avos(X,Y) :- maeE(X,Z), maeE(Z,Y).
```

Como pode ser visto, a compilação do axioma avos gerou quatro novos axiomas, definidos somente sobre os predicados do BDE.

5.5 - Listagem do BDD

Este procedimento foi implementado visando auxiliar o usuário na formulação de consultas sobre o BDD corrente. Seu esquema de operações é bastante simples:

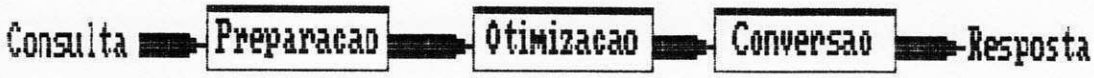
- seleciona a área de código que contém o BDD ("current_world");
- apresenta, na janela de trabalho, todos os axiomas encontrados ("listing").

Na figura abaixo pode ser visto o sistema listando um BDD:

```
H - Otimizador Semantico de Consultas baseado em Heuristicas (OSH)  Versao 2.0
 caRrega   Edita   comPila   Lista   Consulta   Dir   dOs   Fim
 Lista o BDD corrente

 Area de Trabalho
    empE(A,B,C,D).

works_dir_for(A,B) :-
    employee(C,A,D,E),
    dept(E,F,G),
    employee(G,B,H,I).

deptE(dep,function,mngr).

integrity_constraint(gi,[A = B <--,<- empE(A,C,D,E),<- empE(B,C,F,G)]).
integrity_constraint(gi,[(A = B , C = D , E = F) <--,<- empE(G,A,E,C),<- empE(
G,B,F,D)]).
integrity_constraint(gi,[(A = B , C = D) <--,<- deptE(E,A,C),<- deptE(E,B,D)])
.

 Mensagem
--> Listando <--
```

5.6 - Consulta ao BDD

Como foi apresentado anteriormente, este procedimento (consulta) está subdividido em três operações básicas, como pode ser visto na figura abaixo :



Veremos, agora, uma descrição detalhada da implementação de cada operação mostrada acima.

5.6.1 - Preparação da consulta

Este processo é responsável pela recepção da consulta Prolog e pela preparação da consulta para que ela possa ser otimizada. Compõe-se dos seguintes passos:

- lê, na janela do trabalho, a consulta do usuário ("read_string");
- localiza, no string lido, as variáveis resposta (X, Y, Z, W, K) ("pega_variaveis");
- monta um axioma resposta, cujo antecedente é formado pela consulta do usuário convertida de string para termo, e cujo consequente é formado pelo predicado resposta, tendo as variáveis respostas como argumento ("faz_resposta");
- grava o axioma resposta gerado (asserta);
- compila o axioma resposta, utilizando o mesmo procedimento da compilação do BDI ("compila"), podendo gerar vários axiomas respostas;

Como exemplo, vejamos o processo de preparação para a consulta: "Qual a matrícula dos empregados que ganham mais de 5000?"

```
:- empregado(X,_,S,_), S > 5000.
```

Inicialmente, é localizada a variável resposta "X" e em seguida é montado o axioma resposta, seguindo os passos apresentados:

```
resposta(X) :-
        empregado(X,_,S,_),
        S > 5000.
```

Compilando este axioma, o predicado do BDI "empregado" é substituído pelo predicado "empE" do BDE, gerando o seguinte axioma resposta compilado:

```
resposta(X) :-
        empE(X,_,S,_),
        S > 5000.
```

5.6.2 - Otimização da consulta

Este processo realiza as otimizações semânticas sobre os axiomas respostas montados no passo anterior ("otimização"). Ele é composto dos seguintes passos:

- Converte a consulta expressa no axioma resposta para uma expressão em lógica de primeira ordem. É produzida uma lista ("lista_da_consulta"),

cujos elementos são os antecedentes, identificados por "<-", correspondendo aos predicados da consulta do usuário, e o consequente, identificado por "<--", composto pelo predicado resposta ("obtem_axiomas");

- Substitui todas as variáveis na lista, produzida no passo anterior, por constantes distintas, para permitir a instanciação das variáveis nas restrições de integridade. Este processo é feito decompondo os termos da lista até que se encontre uma variável, que é então substituída por uma constante k(n) gerando a lista instanciada da consulta ("aterra_lista_termos").

- Monta uma lista com todas as restrições de integridade do BDD corrente ("findall");

- Os passos seguintes são repetidos para cada RI da lista montada no passo anterior:

    . Separa os antecedentes e os conseqüentes da RI, identificados por "<-" e por "<--", montando duas listas separadas ("ante_cons");

    . Verifica se os antecedentes podem ser inferidos a partir da consulta. Para cada elemento da lista de antecedentes que seja uma referência ao BDE, verifica se este

predicado é membro da lista que representa a consulta. Caso seja, remove o predicado correspondente da consulta e tenta provar o próximo elemento da lista de conseqüentes. Caso o elemento seja um predicado de comparação, verifica se a desigualdade que compõe o predicado pode ser inferida de alguma desigualdade na consulta ("verifica_ante"). Caso algum antecedente não possa ser inferido da consulta, passa para a próxima RI, voltando ao passo anterior;

. Separa as modificações propostas pelos consequentes da RI, devidamente instanciadas no passo anterior. Três classes de modificações são identificadas: igualdades (k(n)=k(m)), desigualdades correspondendo às seleções, e referências a predicados do BDE correspondendo às junções ("separe");

. Cada classe de modificações é então analisada por um conjunto especifico de heurísticas, que determinarão quais as modificações devem ser realizadas sobre a consulta ("modificacoes");

. As modificações que forem consideradas

válidas pelas heuristicas são realizadas sobre a consulta e uma nova RI é selecionada para tratamento;

- Quando não existirem mais RIs aplicáveis, substitui as constantes k(n), na lista instanciada da consulta, por variáveis, registrando as substituições para repeti-las para as outras ocorrências da mesma variável (desaterra_lista_termos);

- Converte a lista da consulta para um novo predicado resposta já otimizado, e grava este novo predicado.

Finalmente serão apresentadas as técnicas de implementação das heuristicas e como elas atuam sobre as classes das modificações propostas.

Para a classe das igualdades (k(n) = k(m)), que aparecem como consequentes das RIs que expressam dependencias funcionais e dependências de chave, existe a heurística H1 (substituição de variáveis), que determina que tais substituições sejam feitas, visando uma possível eliminação de elementos iguais na lista que representa a consulta ("modificacoes_troca" e "sem_duplicatas").

Vejamos, como exemplo, como é tratada a seguinte consulta: "Quais os nomes dos empregados que trabalham

diretamente para João e que ganham mais de 2000 ?"

```
:- trabalha_dir_para(X,joao),
   empregado(_,X,S,_),
   S > 2000.
```

Inicialmente é montado o seguinte predicado resposta:

```
resposta(X) :-
   empE(A,X,B,C),
   deptE(C,D,E),
   empE(E,joao,F,G),
   empE(H,X,I,J),
   I > 2000.
```

que gera a seguinte lista da consulta instanciada:

```
[ resposta(k(1))<-,
  <--empE(k(2),k(1),k(3),k(4)),
  <--deptE(k(4),k(5),k(6)),
  <--empE(k(6),joao,k(7),k(8)),
  <--empE(k(9),k(1),k(10),k(11)),
  <--k(10) > 2000 ]
```

Utilizando a RI1 sobre o primeiro e o quarto elementos na lista da consulta instanciada, nós teríamos como consequente a seguinte igualdade k(9) = k(2). A aplicação desta substituição torna possível a utilização da RI3, obtendo como consequentes as seguintes desigualdades: k(1) = k(1), k(3) = k(10) e

k(4) = k(11). Realizando estas substituições, as duas linhas ficam idênticas, permitindo a eliminação de uma delas, trazendo uma melhora de desempenho na execução da consulta.

A lista da consulta instanciada, já otimizada, fica:

```
[ resposta(k(1))<-,
  <--empE(k(2),k(1),k(3),k(4)),
  <--deptE(k(4),k(5),k(6)),
  <--empE(k(6),joao,k(7),k(8)),
  <--k(3) > 2000 ]
```

Para a classe das desigualdades que aparecem como consequentes, nas RIs representando limites para valores e informações genéricas sobre o BDD, existem três heurísticas que podem ser aplicadas ("modificacoes_selecao").

A primeira é a H2 (eliminação de seleção), onde a desigualdade no consequente da RI é utilizada para eliminar uma seleção redundante que esteja na consulta. A seleção é designada como redundante, se ela puder ser inferida a partir do consequente da RI ("verifica_delecao_selecao").

Como exemplo, vejamos a seguinte consulta formulada sobre o nosso BDD exemplo: "Qual o nome dos empregados que ganham mais de 200 ?"

```
:- empregado(_,X,S,_), S > 200.
```

gerando a seguinte lista da consulta instanciada:

```
[ resposta(k(1))<-,
  <--empE(k(2),k(1),k(3),k(4)),
  <--k(3) > 200 ]
```

Pela restrição de integridade RI7 (limites para valores), nós sabemos que todos os salários dos empregados são maiores que 1000 e menores que 10000, portanto a seleção "S > 200" na lista da consulta é redundante, podendo ser eliminada, reduzindo a lista da consulta otimizada a:

```
[ resposta(k(1))<-,
  <--empE(k(2),k(1),k(3),k(4)) ]
```

A segunda heurística que pode ser aplicada sobre as desigualdades é a H3 (detecção de contradição). É verificado se a desigualdade no consequente da RI determina uma contradição sobre alguma desigualdade na consulta. Neste caso, a consulta é encerrada com uma mensagem de resposta vazia ("verifica_limites").

Vejamos como exemplo a seguinte consulta: "Qual o nome dos empregados que ganham menos de 500 ?"

```
:- empE(_,X,S,_), S < 500.
```

Pela RI7 (limites para valores) nós sabemos que

todos os empregados ganham mais de 1000, portanto não pode existir nenhum empregado que ganhe menos de 500, logo a resposta para a consulta é vazia.

A terceira heurística que se aplica às desigualdades é a H4 (introdução de seleção). Ela determina que uma desigualdade expressa no consequente de uma RI pode ser incluída na consulta, caso exista um índice pelo atributo restringido na desigualdade. A melhora no desempenho é obtida pela substituição de uma pesquisa sequencial sobre a relação por uma pesquisa indexada sobre o atributo da desigualdade ("verifica_inclusao_selecao" e "verifica_indice").

Como exemplo, vejamos a consulta: "Qual a matrícula dos empregados que trabalham no departamento 04 ?"

```
:- empE(X,_,G,04).
```

A restrição de integridade RI9 (restrição semântica) determina que todos os empregados do departamento 04 ganham mais do que 8000. Supondo a existência de um índice por salário na relação empE, a seleção "G > 8000" será incluída na consulta, melhorando substancialmente o tempo de resposta.

A terceira e última classe das modificações é composta por aquelas cujo consequente da RI especificam junções (referências aos predicados do BDE). Nesta

classe, estão as integridades referenciais e as informações genéricas, existindo basicamente duas heurísticas que determinam as modificações ("modificacao_juncao").

A primeira é a H5 (eliminação de junções) que determina que o consequente de uma RI, sendo uma referência a um predicado do BDE, pode ser eliminado da consulta, desde que seja obedecida a regra apresentada na seção 3.3. Ou seja, as variáveis no consequente da RI que não participam da junção não podem aparecer em outro lugar na consulta ("verifica_lista_termos").

Vejamos a seguinte consulta: "Qual o nome dos empregados que ganham mais de 2000 e que estão alocados a um departamento ?"

```
:- empE(_,X,S,D), S > 2000, deptE(D,F,G)
```

Pela restrição de integridade RI5 (integridade referencial), nós sabemos que todos os empregados estão alocados em departamentos, e como as variáveis "F" e "G" não aparecem em nenhum outro lugar da consulta, a junção pode ser eliminada, trazendo uma melhoria significativa para o desempenho da consulta.

A segunda e última heurística que determina modificações sobre junções especificadas nos consequentes da RI é a H6 (inclusão de junção). Ela determina que uma junção pode ser incluída na consulta

original, desde que exista um índice agrupado para as duas relações, pelo atributo de junção ("verifica_indice_agrupado")

Como exemplo, supondo a existência de um índice agrupado pelo atributo "dep" em empE e deptE, para a consulta: "Qual o nome dos empregados que ganham mais de 8000 ?", nós temos:

```
:- empE(_,X,S,_), S > 8000.
```

Pela RI8 (restrição semântica), nós sabemos que todo empregado que ganha mais do que 8000 trabalha na presidência e joga golf. Verificamos que duas junções podem ser introduzidas na consulta (com deptE e com especE). Contudo, apenas uma (com deptE) satisfaz a heurística H6. A inclusão desta junção substitui uma pesquisa sequencial sobre empE por uma pesquisa indexada pelo atributo "dep", trazendo uma melhora no desempenho da consulta.

5.5.3 - Conversão da consulta

Este processo é responsável pela conversão dos axiomas resposta para SQL, e pela submissão da consulta ao SGBD relacional, apresentando os resultados ao usuário. O processo de conversão é similar ao descrito por Jarke [JARK84] e é composto basicamente dos seguintes passos:

- Converte o axioma resposta para uma lista com antecedentes e consequentes em lógica de primeira ordem (igual ao primeiro passo da otimização);

- Substitui as variáveis por constantes k(n) (igual ao segundo passo da otimização), gerando uma nova lista instanciada da consulta;

- Para cada elemento da lista consulta não instanciado, que seja um predicado de referência ao BDE, monta a lista atribuída da consulta do seguinte modo:

    . Substitui as variáveis pelos nomes dos atributos correspondentes, obtidos dos esquemas das relações do BDE ("aterra_esquema");

    . Concatena, a cada nome de atributo obtido no passo anterior, um identificador de relação composto por uma letra "v" junto com o número sequencial que identifica esta relação ("renomeia_argumentos");

    . Coloca o nome da relação e o número sequencial dado para ela em uma terceira lista que chamaremos de lista das relações;

- Monta a cláusula FROM da consulta utilizando a

lista de relações produzida no passo anterior ("clausula_from");

- Monta uma quarta lista (lista de variáveis), a partir da lista instanciada da consulta com constantes k(n) e da lista atribuída da consulta com os atributos, onde cada elemento da nova lista associa uma constante k(n) a um nome de atributo ("coleciona_atributo");

- Classifica esta lista, para que todas as referencias à mesma variável fiquem juntas ("sort");

- Monta a cláusula SELECT a partir do predicado resposta, que é o primeiro elemento da lista instanciada da consulta. Cada argumento do predicado deve ser localizado na lista de variáveis para determinar o nome exato do atributo ("clausula_select");

- A geração da cláusula WHERE esta subdividida em três passos:

    . Para cada par de variáveis iguais encontrados na lista de variáveis, monta-se uma equi-junção com os nomes dos atributos correspondentes ("juncoes_where");

    . Para cada elemento atômico da lista de variáveis, constituindo uma constante,

monta-se uma igualdade (=) onde o nome do atributo é retirado da própria lista ("igualdades_where");

. Para cada predicado de comparação na lista da consulta instanciada, substitui as constantes k(n) pelos nomes dos atributos correspondentes usando a lista de variáveis ("desigualdades_where");

- Concatena as cláusulas SELECT, FROM e WHERE montadas nos passos anteriores gerando o comando SQL;

- Submete o comando gerado ao SGBD e apresenta as respostas da consulta ao usuário.

Vejamos um exemplo para ilustrar o funcionamento deste processo. Imaginemos o axioma resposta para a seguinte consulta: "Quais os nomes dos empregados da presidência que ganham mais de 9000 ?"

```
reposta(X) :-
     empE(M,X,S,D),
     deptE(D,presidencia,C),
     S > 9000.
```

Montando inicialmente a lista, nós temos:

```
[ resposta(X)<-,
 <--empE(M,X,S,D),
```

```
<--deptE(D,presidencia,C),
<--S > 9000<-- ]
```

Seguindo o processo, e substituindo as variáveis por constantes, nós obtemos a seguinte lista instanciada da consulta:

```
[ resposta(k(1))<-,
 <--empE(k(2),k(1),k(3),k(4)),
 <--deptE(k(4),presidencia,k(5)),
 <--k(3) > 9000 ]
```

Montando a lista atribuida da consulta, nós temos:

```
[ resposta(X)<-,
  <--empE(v1.matr,v1.nom,v1.sal,v1.dep),
  <--deptE(v2.dep,v2.fun,v2.chf),
  <--S > 9000 ]
```

A lista de relações fica:

```
[ (1, empE), (2, deptE) ]
```

Neste ponto, é montada a cláusula FROM da consulta, com base na lista de relações:

```
FROM empE v1, deptE v2
```

Em seguida, é construida a lista classificada de variáveis que ficaria:

```
[ (k(1), v1.nom), (k(2), v1.matr),
```

```
(k(3), v1.sal), (k(4), v1.dep),
(k(4), v2.dep), (k(5), v2.chf),
(presidencia, v2.fun) ]
```

Neste ponto, é montada a cláusula SELECT, analisando o predicado resposta na lista da consulta instanciada e localizando os nomes dos atributos na lista de variáveis. No nosso caso, seria gerado:

```
SELECT v1.nom
```

Para a cláusula WHERE são geradas as junções, a partir dos pares encontrados na lista instanciada da consulta, formando a seguinte cláusula:

```
WHERE (v1.dep = v2.dep)
```

Continuando o processo, são geradas as igualdades para as constantes encontradas na lista de variáveis:

```
WHERE (v1.dep = v2.dep) and
      (v2.fun = "presidencia")
```

Finalmente, são geradas as desigualdades encontradas na lista instanciada da consulta, buscando o nome dos atributos na lista de variáveis:

```
WHERE (v1.dep = v2.dep) and
      (v2.fun = "presidencia") and
      (v1.sal > 9000)
```

Unindo as cláusulas geradas, é produzido o

seguinte comando SQL:

```
SELECT v1.nom
FROM empE v1, deptE v2
WHERE (v1.dep = v2.dep) and
      (v2.fun = "presidencia") and
      (v1.sal > 9000);
```

que é submetido ao SGBD para obtenção das respostas.

Na figura abaixo, nós podemos ver uma demonstração do sistema em funcionamento, processando uma consulta do usuário:

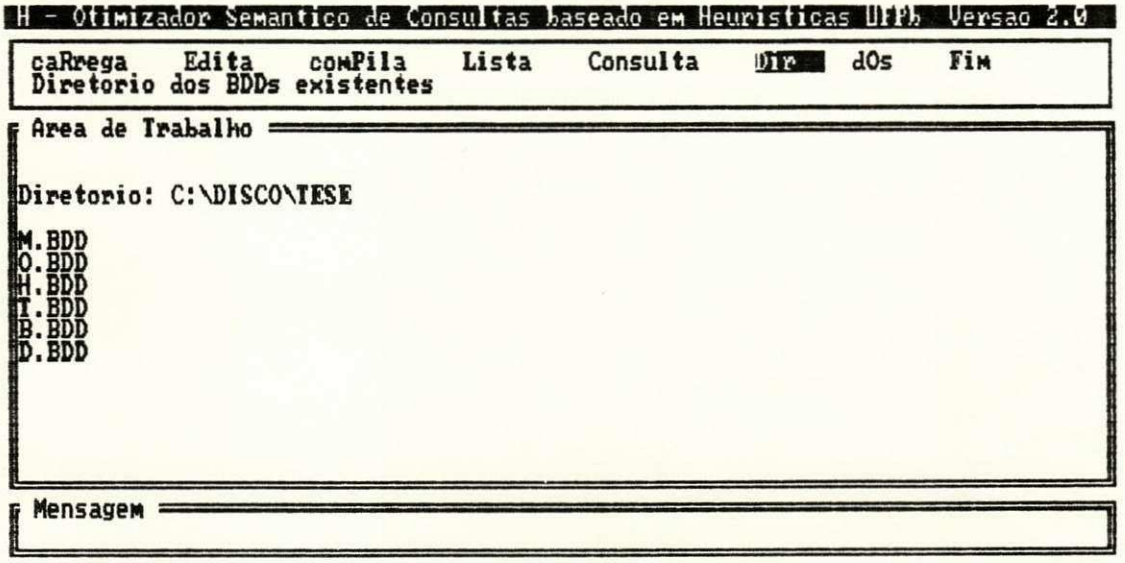
```
H - Otimizador Semantico de Consultas baseado em Heuristicas (OTP)  Versao 2.0
 caRrega   Edita   coMPila   Lista   Consulta   Dir   dOs   Fim
 Consulta o BDD corrente
 Area de Trabalho
 Consulta --> employee(_,X,S,_), S > 2000

 resposta(A) :-
     empE(B,A,C,D),
     C > 2000.

 Rem  Bdd: h
 Rem Data: 1/11/1989
 Rem Hora: 16:12:46

 select v1.name
 from empE v1
 where (v1.salary > 2000);

 Mensagem
```

5.7 - Diretório dos BDDs

Este módulo da interface permite ao usuário verificar quais os BDDs existentes no diretório corrente que podem ser carregados no sistema ("directory").

Podemos ver na figura abaixo o sistema apresentando os

BDDs existentes no diretório corrente:



5.8 - Interface com o DOS

Neste módulo, é permitido ao usuário voltar ao sistema operacional temporariamente, para realizar qualquer tarefa desejada. O retorno para o sistema é feito através do comando EXIT ("shell").

5.9 - Término das operações

Esta opção da interface permite ao usuário sair do sistema, retornando o controle, definitivamente, para o sistema operacional ("halt").

VI - Avaliação e Exemplos

Neste capítulo, é apresentada uma avaliação de desempenho do processo de otimização, visando uma análise das características e capacidades do otimizador. Inicialmente, é feito um levantamento dos elementos que determinam o desempenho do processo de otimização e, posteriormente, são apresentados alguns exemplos que demonstram a capacidade do otimizador em reduzir o tempo de resposta das consultas submetidas pelos usuários. Os exemplos forão retirados dos principais trabalhos da área para que pudéssemos avaliar comparativamente as capacidades da nossa implementação do otimizador.

6.1 - Análise de desempenho do processo de Otimização

Em uma rápida análise sobre o processo de otimização, podemos destacar os seguintes componentes que determinam o desempenho do otimizador:

- número de RIs do BDD;
- número de RIs aplicáveis sobre a consulta;
- número de cláusulas da consulta compilada.

O primeiro componente determina o espaço de pesquisa, sobre o qual são selecionadas as RIs aplicáveis, de acordo com as heurísticas apresentadas. Esta pesquisa é realizada sobre todas as RIs, e ocorre quando uma modificação é introduzida na consulta original, produzindo uma nova consulta. Portanto o espaço de pesquisa influencia diretamente no desempenho do otimizador, mas que seu peso

na análise geral depende exclusivamente do segundo componente, que determina quantas vezes será realizada esta pesquisa.

Contudo, o número de RIs aplicáveis sofre uma grande redução, em relação ao número geral de RIs, devido às heurísticas, que restringem a aplicabilidade das RIs, reduzindo a complexidade geral do processo. O terceiro componente influencia indiretamente o desempenho do processo, já que quanto mais cláusulas existirem na consulta, maior será a possibilidade de que uma RI seja aplicável sobre ela.

Concluindo, podemos perceber que o fator determinante do desempenho do otimizador é o número de RIs aplicáveis, que determina as modificações sobre as consultas, garantindo, pelas heurísticas, uma melhora do desempenho na execução da consulta.

6.2 - Exemplos de Otimização

A seguir, apresentamos exemplos que refletem alguns resultados positivos, obtidos pelo processo de otimização. Os tempos apresentados correspondem ao tempo de execução das consultas. Os tempos utilizados pelo sistema para realizar as otimizações nunca foram superiores a três segundos.

Ambiente:

Tabelas: empE, deptE, especE.

Cardinalidades: card(empE) = 2000.

card(deptE) = 4.

card(especE) = 2000.

Indice Agrupado: Atributo Dep em empE e deptE.

a. Consulta : empregado(_,X,S,_),S > 8000

a.1 Sem Otimização Semântica - Tempo: 18 Segundos

"Quais os nomes dos empregados que ganham mais de 8000 ?"

```
resposta(A):-
     empE(B,A,C,D),                    (Preparação)
     C > 8000.

select v1.nome
from empE v1                           (Conversão)
where (v1.salario > 8000);
```

a.2 Com Otimização Semântica - Tempo: 06 Segundos

Heurísticas Utilizadas: Introdução de Junções

"Quais os nomes dos empregados que trabalham na presidência e ganham mais de 8000 ?"

```
resposta(A) :-
     empE(B,A,C,D),                    (Preparação)
                                            e
     C > 8000,                         (Otimização)
     deptE(D,presidencia,E).

select v1.nome
from empE v1, deptE v2
```

```
where (v1.dep = v2.dep) and
      (v2.funcao = 'presidencia') and
      (v1.salario > 8000);
```

b. Consulta: mesmo_gerente(X,jose),empregado(_,X,S,_),S>8000

b.1 Sem Otimização Semântica - Tempo: Infinito

"Quais os nomes dos empregados que possuem o mesmo gerente que jose e que ganham mais de 8000 ?"

```
resposta(A) :-
     empE(B,A,C,D),
     deptE(D,E,F),
     empE(F,G,H,I),
     empE(J,jose,K,L),
     deptE(L,M,N),
     empE(N,G,O,P),
     A \= jose,
     empE(Q,A,R,S),
     R > 8000.

select v1.nome
from empE v1, deptE v2, empE v3, empE v4,
     deptE v5, empE v6, empE v7
where (v1.nome = v7.nome) and
      (v1.dep = v2.dep) and
      (v2.chefe =  v3.matr) and
      (v3.nome = v6.nome) and
      (v4.dep = v5.dep) and
```

```
        (v5.chefe = v6.matr) and
        (v4.nome = 'jose') and
        (not v1.nome = 'jose') and
        (v7.salario > 8000);
```

b.2 Com Otimização Semântica - Tempo: 16 Segundos

Heuristicas Utilizadas: Substituição de Variáveis
Eliminação de Junções
Introdução de Junções

"Quais os nomes dos empregados que trabalham no mesmo departamento que jose e ganham mais de 8000, sabendo-se que jose trabalha na presidencia ?"

```
resposta(A) :-
     empE(B,jose,C,D),
     A \= jose,
     empE(E,A,F,D),
     F > 8000,
     deptE(D,presidencia,G).

select v2.nome
from empE v1, empE v2, deptE v3
where (v1.dep = v2.dep) and
      (v2.dep = v3.dep) and
      (v1.nome = 'jose') and
      (v3.funcao = 'presidencia') and
      (not v2.nome = 'jose') and
      (v2.salario > 8000);
```

c. Consulta: trabalha_dir_para(X,X),empregado(_,X,S,_),S>8000

c.1 Sem Otimização Semântica - Tempo: Infinito

"Quais os nomes dos empregados que trabalham diretamente para si mesmos e que ganham mais que 8000 ?"

```
resposta(A) :-
     empE(B,A,C,D),
     deptE(D,E,F),
     empE(F,A,G,H),
     empE(I,A,J,K),
     J > 8000.
```

```
select v1.nome
from empE v1, deptE v2, empE v3, empE v4
where (v1.nome = v3.nome) and
      (v3.nome = v4.nome) and
      (v1.dep = v2.dep) and
      (v2.chefe = v3.matr) and
      (v4.salario > 8000);
```

c.2 Com Otimização Semântica - Tempo: 02 Segundos

Heurísticas Utilizadas: Substituição de Variáveis
Eliminação de Junções
Introdução de Junções

"Quais os nomes dos empregados da presidencia que

ganham mais de 8000 ?"

```
resposta(A) :-
     empE(B,A,C,D),
     C > 8000,
     deptE(D,presidencia,B).
```

```
select v1.nome from empE v1, deptE v2
where (v1.matr = v2.chefe) and
      (v1.dep = v2.dep) and
      (v2.funcao = 'presidencia') and
      (v1.salario > 8000);
```

d. Consulta: empregado(_,X,S,_),S > 11000

d.1 Sem Otimização Semântica - Tempo: 10 Segundos

"Quais os nomes dos empregados que ganham mais de 11000"

```
resposta(A) :-
     empE(B,A,C,D),
     C > 11000.
```

```
select v1.nome
from empE v1
where (v1.salario > 11000);
```

d.2 Com Otimização Semântica - Tempo: 00 Segundos

Heurísticas Utilizadas: Detecção de Contradição

Mensagem indicando resposta nula

VII - Conclusões e Trabalhos Futuros

Neste trabalho, descrevemos o projeto e a implementação de um otimizador semântico de consultas a bancos de dados dedutivos, baseado em heurísticas, que selecionam as modificações sugeridas pelas informações semânticas, expressas nas restrições de integridade. Estas restrições podem ser de várias categorias e estão representadas em lógica de primeira ordem.

Quanto à operacionalização do sistema, nele destacamos duas qualidades importantes:

- o aproveitamento integral da capacidade dos SGBDs já existentes, os quais são cruciais para a gerência de grandes bases de fatos (BDE);

- a independência do sistema de um SGBD particular (como, por exemplo, o Oracle), uma vez que a interface SQL gerada é padrão para bancos de dados relacionais.

Em relação à aplicabilidade, o sistema pode ser utilizado em diversos tipos de projetos de inteligência artificial que possuam grandes bases de fatos, como, por exemplo, em sistemas especialistas de grande porte.

A partir deste trabalho podem ser realizadas algumas pesquisas complementares:

- Adaptação dos mecanismos de compilação e otimização de consultas utilizados neste trabalho para que possam tratar axiomas do BDI que possuam definição recursiva. Alguns

trabalhos como [LEE88] e [VICI86] devem ser analisados, para determinar a abordagem a ser utilizada;

- Aperfeiçoamento do Otimizador para tratamento de RIs e consultas mais genéricas;

- Apresentação pelo Otimizador dos passos intermediários do processo de Otimização;

- Armazenamento das RIs e das regras de dedução do BDI no SGBD relacional, viabilizando a construção de grandes BDDs [RAMN88];

- Desenvolvimento de novas heuristicas e aperfeiçoamento daquelas já utilizadas pelo otimizador semântico, visando melhorar ainda mais os resultados iniciais obtidos;

- Implementação do Otimizador utilizando uma linguagem de programação convencional visando uma análise comparativa de desempenho.

VIII - Bibliografia

[BICA90] Bicalho,J.A.N., Sampaio,M.C. "Uma abordagem heuristica para a otimização semântica de consultas a bancos de dados dedutivos", Anais do V Simpósio Brasileiro de Banco de dados, 216-230, Rio de Janeiro, Abril 1990.

[CHAK86] Chakravarthy,U.S., Fishman, D., Minker, J. "Semantic Query Optimization in Expert Systems and Database Systems", em [KERS86], 659-674.

[CHAN73] Chang,C.L., Lee,R.C.T. Symbolic Logic and Mechanical Theorem Proving, Academic Press, New York, 1973.

[CHAN86] Chang, C.L., Walker, A., "PROSQL: A Prolog Interface with SQL/DS", em [KERS86].

[DATE85] Date, C.J. A Guide to DB2, Addison Wesley, 1985.

[GALL84] Gallaire H., Minker J., Nicolas J. M. "Logic and Database: A Deductive Approach", ACM Computing Surveys, 153-185, vol 16, num 2, junho 1984.

[GHOS88] Ghosh, S., Lin, C.C., Sellis, T., "Implementation of a Prolog-INGRES Interface", Sigmod Record, 17, 2, June 1988.

[JARK84] Jarke,M., "An Optimizing Prolog Front-End to a Relational Query System", Proceedings of ACM-SIGMOD Conference, Boston, 1984, 296-306.

[JARK86] Jarke,M., "External Query Simplification: A Graph Theoretic Approach and its Implementation in Prolog" em [KERS86], 675-692.

[KERS86] Kerschberg,L. (ed) Expert Database Systems, Proceedings of First International Workshop, Benjamin/Cummings Inc., Menlo Park, CA 1986.

[KING81] King,J.,J. "Quist: a system for semantic query optimization in relational databases", Proc. of 7a International Conference on Very Large Databases, 510-517, IEEE, Cannes, France, September 1981.

[LEE88] Lee, S., Han,J. "Semantic Query Optimization in Recursive Databases", Proc. of 4a International Conference on Data Engineering, 444-451, IEEE, 1988

[LLOY85] Lloyd, J. W., Topor, R. W. "A Basis for Deductive Database Systems", Journal of Logic Programming, 1985, 2, 93-109.

[MINK88] Minker, J., Lobo, J. "A Metaprogramming Approach to Semanticaly Optimize Queries in Deductive Databases", Expert Database Systems, 1988, 699-741.

[ORAC84] Oracle Reference Manual, Oracle Corporation,Menlo Park, California, 1984.

[RAMN88] Ramnarayan, R., Lu, H. "A Data/Knowledge Base Management Testbed and Experimental Results on Data/Knowledge Base Query and Updade Processing", Proceedings of ACM SIGMOD Conference on Management of Data, Chicago, 17, 3, 387-395, 1988.

[SHEN87] Shenoy, S. T., Ozsoyoglu, Z. M. "A System for Semantic Query Optimization", Proceedings of ACM SIGMOD Conference, 1987, 181-195.

[ULLM82] Ullman, J. D. Principles of Databases Systems, 2 ed, Computer Science Press., Potomac, Md., 1982

[VICI86] Vicille, L. "Recursive Axioms in Deductive Databases: The Query/Subquery Approach", Proceedings of the First International Conference on Expert Database Systems, Charleston, South Caroline, USA, 179-194, April 1986.