UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CPREF-SQL: UMA LINGUAGEM DE CONSULTA COM SUPORTE A PREFERÊNCIAS CONDICIONAIS - TEORIA E IMPLEMENTAÇÃO

FABÍOLA SOUZA FERNANDES PEREIRA

Uberlândia - Minas Gerais
2011

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FABÍOLA SOUZA FERNANDES PEREIRA

CPREF-SQL: UMA LINGUAGEM DE CONSULTA COM SUPORTE A PREFERÊNCIAS CONDICIONAIS - TEORIA E IMPLEMENTAÇÃO

Dissertação de Mestrado apresentada à Faculdade de Ciência da Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Banco de Dados.

Orientadora:
Prof[a]. Dr[a]. Sandra Aparecida de Amo

Uberlândia, Minas Gerais
2011

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Ciência da Computação a aceitação da dissertação intitulada "**CPref-SQL: uma Linguagem de Consulta com Suporte a Preferências Condicionais - Teoria e Implementação**" por **Fabíola Souza Fernandes Pereira** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 16 de Maio de 2011

Orientadora: _____

Profª. Drª. Sandra Aparecida de Amo
Universidade Federal de Uberlândia

Banca Examinadora:

_____

Prof. Dr. Caetano Traina Júnior
Universidade de São Paulo

_____

Prof. Dr. Ilmério Reis da Silva
Universidade Federal de Uberlândia

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Data: Maio de 2011

Autor:	**Fabíola Souza Fernandes Pereira**
Título:	**CPref-SQL: uma Linguagem de Consulta com Suporte a Preferências Condicionais - Teoria e Implementação**
Faculdade:	**Faculdade de Ciência da Computação**
Grau:	**Mestrado**

_____

Autor

Dedicatória

Aos meus pais, Iron e Solimar e à minha irmã Flávia.

A meu amado Diogo Nunes.

Agradecimentos

A Deus,
pois sem Ele nada seria possível.

À Profª. Sandra,
minha orientadora, pelas oportunidades e desafios que me proporcionou, por seu apoio, incentivo, inúmeros conselhos em todos os momentos e, principalmente, por sua confiança. A você, Sandra, minha eterna gratidão.

Aos meus pais, Iron e Solimar,
pelo amor, apoio e estímulo dispensados incondicionalmente durante toda a minha vida. Por entenderem minha ausência nos aniversários e feriados. Por vibrarem com cada uma de minhas conquistas, grandes ou pequenas. A vocês, dedico todo o meu esforço.

À minha irmã Flávia,
por tornar meus dias mais alegres e engraçados, mesmo que a distância.

A meu amado Diogo Nunes,
por estar sempre ao meu lado e compreender minha ausência em tantos finais de semana. Seu abraço e palavras carinhosas faz tudo parecer muito mais fácil.

Aos amigos de laboratório,
pela convivência intensa, pelos almoços, risadas, conselhos e muitas piadas que fizemos. Em especial, à Nádia (por ouvir meus desabafos!), Gabriel, Felipe, Vinícius Ruela, Guilherme, Glauco e Murilo.

Aos graduandos Gildo e Vinícius,
fundamentais no meu trabalho. Obrigada pela disponibilidade e excelência em me ajudarem nas atividades extras.

Ao pessoal do GBDI de São Carlos.
Foi uma experiência muito prazerosa. Em especial, ao Prof. Caetano e Profª. Agma pela oportunidade e à Mônica pelo esforço, amizade e disposição em realizarmos o trabalho conjunto.

Aos amigos do grupo de estudos FINLAN.
Cresci muito com nossas reuniões, artigos e, principalmente, experiência trocada nesses anos de convivência.

"Se enxerguei mais longe, foi porque me apoiei sobre ombros de gigantes."
(Isaac Newton)

Resumo

Muitas aplicações importantes como comércio eletrônico e sistemas de recomendação, demandam o uso de técnicas eficientes de personalização e manipulação de preferências do usuário. No campo de banco de dados, existem diversas pesquisas voltadas para a extensão da linguagem SQL padrão com o intuito de desenvolver linguagens de consulta capazes de expressar e filtrar preferências. De fato, sistemas de banco de dados que integram preferências do usuário têm sido uma boa solução para personalização e alta qualidade nas respostas a consultas.

Nesta dissertação propomos a implementação da linguagem de consulta CPref-SQL, que é uma extensão SQL com suporte a preferências condicionais, na qual as preferências são especificadas por um conjunto de regras expressas de acordo com um formalismo lógico.

Para tanto, generalizamos o modelo de preferência da linguagem CPref-SQL, tornando-o mais expressivo e desenvolvemos os algoritmos G-BNL e GRank-BNL que implementam os novos operadores de preferência ***Select-Best*** e ***SelectK-Best***, respectivamente. Esses operadores são capazes de avaliar consultas top-k com preferências, ou seja, consultas que retornam as K tuplas mais preferidas de acordo com uma hierarquia de preferências do usuário. Por fim, implementamos de fato a linguagem, no *core* do processador de consultas do PostgreSQL. Um extenso conjunto de testes demonstra a eficiência dos métodos utilizados no desenvolvimento da linguagem.

Diante da implementação, apresentamos, ainda, estudos de caso que ilustram a aplicação da linguagem CPref-SQL num contexto de integração entre consultas por similaridade e preferências do usuário.

Palavras chave: preferências condicionais, consultas com preferências, extensão sql, consultas top-k

Abstract

Many important applications like e-commerce and recommendation systems require the use of efficient techniques for customizing and manipulating user preferences. In the database field, the researches are focused on the development of query languages able to express and filter preferences. In fact, database systems that integrate user preferences have been a good solution for personalization and high quality in queries answers.

In this dissertation we propose the implementation of the CPref-SQL query language, an SQL extension that supports conditional preferences. In this language, the preferences are specified by a set of rules expressed in terms of a logical formalism.

To this end, we generalize the CPref-SQL preference model, making it more expressive and develop the algorithms G-BNL and GRank-BNL, which implement the new preference operators ***Select-Best*** and ***SelectK-Best***, respectively. These operators are able to evaluate top-k queries with preferences, i.e., queries that return the K most preferred tuples according to a preference hierarchy. Finally, we implemented the language in the core of the PostgreSQL query processor. An extensive set of experiments demonstrates the efficiency of the methods used in language development.

Given the implementation, we also present case studies that illustrate the application of the CPref-SQL language in a context of integration between similarity queries and user preferences.

Keywords: conditional preferences, preference queries, sql extension, top-k queries

Sumário

Lista de Figuras **xxi**

Lista de Tabelas **xxiii**

1 Introdução **25**
1.1 Motivação 27
1.2 Contribuições 28
1.3 Organização da dissertação 29

2 Trabalhos Correlatos e Fundamentos Teóricos **31**
2.1 Raciocínio e Modelagem com Preferências 31
2.1.1 CP-nets e TCP-nets 32
2.1.2 Linguagem de Preferências Condicionais 33
2.1.3 *Frameworks* em Banco de Dados 37
2.2 Operadores, Algoritmos e Extensões SQL para Avaliação de Preferências . 38
2.2.1 O Operador *Skyline* 38
2.2.2 O Operador *Winnow* 40
2.2.3 Preference SQL 41
2.3 Consultas Top-K 42
2.4 Implementação de Módulos de Preferência 44
2.5 Considerações Finais 46

3 A Linguagem CPref-SQL **47**
3.1 O Modelo de Preferência 48
3.1.1 Teoria de Preferência Condicional 48
3.1.2 Ordem de Preferência 50
3.1.3 Teste de Consistência 53
3.2 CPref-SQL: uma extensão SQL 64
3.2.1 Operadores da Álgebra CPref-SQL 64
3.2.2 Sintaxe CPref-SQL 65
3.2.3 Plano de Execução de uma Consulta-pc Top-K 67
3.3 Considerações Finais 67

4 Algoritmos e Implementação **69**
4.1 Teste de Dominância . . . 69
4.1.1 Ordem Forte . . . 70
4.1.2 Ordem Fraca . . . 72
4.2 Algoritmo G-BNL . . . 75
4.3 Algoritmo GRank-BNL . . . 78
4.4 Implementação . . . 82
4.4.1 CPref-SQL na Arquitetura PostgreSQL . . . 83
4.4.2 Abordagem *On-Top* . . . 85
4.4.3 Organização do Código . . . 88
4.5 Reescrita em SQL . . . 89
4.6 Considerações Finais . . . 90

5 Resultados Experimentais **91**
5.1 Ambiente Experimental . . . 91
5.1.1 Base de Dados . . . 92
5.1.2 Consultas . . . 92
5.1.3 Preferências . . . 93
5.1.4 Tamanho do *Buffer* . . . 93
5.2 Análise de Performance . . . 94
5.3 Análise de Escalabilidade . . . 96
5.4 Implementação *On-Top* . . . 98
5.4.1 Ambiente Experimental . . . 98
5.4.2 Análise de Performance . . . 98
5.4.3 Análise de Escalabilidade . . . 99
5.5 Considerações Finais . . . 100

6 Estudos de Caso **101**
6.1 Similaridade e Preferências sobre Imagens Médicas . . . 101
6.1.1 Módulo de Preferência no SIREN . . . 102
6.1.2 Avaliação Experimental . . . 105
6.1.3 Conclusões . . . 106
6.2 Mineração de Preferências em Consultas por Similaridade . . . 107
6.2.1 Módulo de Mineração de Dados do SIREN . . . 107
6.2.2 Extração de um Modelo de Preferência . . . 109
6.2.3 Resultados Experimentais . . . 112
6.2.4 Conclusões . . . 113
6.3 Considerações Finais . . . 114

7 Conclusão **115**
7.1 Principais Contribuições 115
7.2 Propostas para Trabalhos Futuros 117

Referências Bibliográficas **121**

A A Linguagem CPref-SQL na Web **127**

Lista de Figuras

1.1 Aperfeiçoamento de consultas em SGBDRs 26

2.1 CP-net para *Viagens* e seu grafo de preferência induzido 33
2.2 Grafo de dependência preferencial da teoria-pc Γ. 35
2.3 *Skyline* de Viagens 39
2.4 Uma instância da relação `Viagens` 40
2.5 Resultado da operação $w_{C_1}(Viagens)$. 41
2.6 Funções de ranking da instância de `Viagens` 42
2.7 Abordagens de consultas top-K 44
2.8 Arquitetura FlexPref ([Levandoski et al. 2010b]) 45

3.1 Uma instância da relação `Viagens` 49
3.2 Grafo *better-than* da relação `Viagens` sob a ordem forte de preferência. . . 52
3.3 Grafo de dependência preferencial da teoria-pc Γ. 54
3.4 Grafo de dependência preferencial da teoria-pc Γ. 56
3.5 Grafo *better-than* da relação `Viagens` sob a ordem fraca de preferência. . . 57
3.6 Algoritmo **ConsistencyTest** 60
3.7 Algoritmo **CheckDependencyGraph** 60
3.8 Algoritmo **CheckLocalConsistency** 61
3.9 Grafo de dependência preferencial da teoria-pc Γ. 63
3.10 Catálogo para as preferências *myprefs* 66
3.11 Plano canônico de execução de uma consulta CPref-SQL 67

4.1 Esquema do algoritmo **strongDomTest** 70
4.2 Algoritmo **strongDomTest** 71
4.3 Esquema do algoritmo **weakDomTest** 73
4.4 Algoritmo **weakDomTest** 74
4.5 G(Γ) com programas Datalog associados aos nós R e Du. 74
4.6 Algoritmo **G-BNL** 76
4.7 Grafo *better-than* da relação `Viagens` sob a ordem fraca de preferência. . . 77
4.8 Iterações do algoritmo G-BNL 78
4.9 Algoritmo **GRank-BNL** 79

4.10 Algoritmo **topK** 79
4.11 Grafo *better-than* da relação `Viagens` sob a ordem forte de preferência. . . 80
4.12 Iterações do algoritmo GRank-BNL 82
4.13 Diagrama da arquitetura PostgreSQL. Componentes em azul indicam aqueles modificados para a extensão CPref-SQL 83
4.14 Diagramas de sintaxe dos novos comandos CPref-SQL 84
4.15 Plano canônico de execução de uma consulta CPref-SQL *on-top* 88

5.1 Esquema do TPC-H 92
5.2 Performance e número de tuplas retornadas variando número de regras . . 94
5.3 Performance e número de tuplas retornadas variando nível de profundidade das teorias-pc 95
5.4 (a) Performance variando o buffer. (b) Performance do ***SelectK-Best*** . . . 96
5.5 Escalabilidade e número de tuplas retornadas variando tamanho da base de dados 97
5.6 Escalabilidade e número de tuplas retornadas variando consulta TPC-H . . 97
5.7 Resultados de performance 99
5.8 Resultados de escalabilidade 100

6.1 Arquitetura SIREN com módulo de preferência 103
6.2 Porcentagem de respostas corretas em consultas por similaridade e preferências 106
6.3 Instância da relação `MaravilhasDoMundo` 108
6.4 Precisão das respostas às consultas por similaridade 114

7.1 Novas tendências de consultas em SGBDs 118

A.1 Página inicial do *site* 127
A.2 Interface de demonstração 128
A.3 Interface de demonstração - `CREATE PREFERENCES` 128
A.4 Interface de demonstração - bloco `SELECT` 129

Lista de Tabelas

3.1 Características da linguagem CPref-SQL 47
3.2 Comparação entre as tuplas da relação `Viagens` 57

5.1 Bases de dados utilizadas nos experimentos 94
5.2 Bases de dados utilizadas nos experimentos para versão *on-top* da linguagem 98

Capítulo 1

Introdução

As preferências de um usuário determinam suas escolhas. Nos atuais sistemas de informação, a manipulação de preferências tem sido cada vez mais importante. De fato, há uma necessidade de que sistemas possam resolver perguntas do tipo: qual produto deve ser recomendado a esse cliente que acabou de fazer uma compra? Ou ainda, qual a melhor maneira de ordenar os resultados de uma busca para melhor agradar ao usuário? Comércio eletrônico, sistemas multimídia e busca na web são alguns exemplos de aplicações que lidam diretamente com o desafio de filtrar a informação realmente interessante ao usuário, além de utilizá-la para formular políticas que melhoram e automatizam tomadas de decisão.

Intuitivamente, preferências são desejos pessoais do tipo *"gosto mais de A do que de B"*, que podem ser especificadas de maneira qualitativa ou quantitativa. Por exemplo, dada uma relação de opções de pacotes turísticos, para encontrar as viagens mais interessantes para o usuário de maneira quantitativa, basta solicitar que o mesmo dê uma nota a cada opção de pacote, de maneira que aqueles com maiores notas serão os selecionados. Pela abordagem qualitativa, é possível obter do usuário informações sobre quais atributos afetam sua preferência. Assim, se ele prefere viajar para a praia do que para uma cidade grande, já consegue-se obter uma classe de destinos preferidos sem que tenha sido necessário avaliar cada opção individualmente.

As pesquisas neste tema têm focado no desenvolvimento de formalismos para manipulação de preferências. Na área de Inteligência Artificial, por exemplo, o interesse está em desenvolver mecanismos de especificação e raciocínio com preferências [Boutilier et al. 2004, Brafman et al. 2006, Wilson 2004]. O objetivo é buscar modelos de preferências e a melhor maneira de expressá-los em subáreas como planejamento, robótica e processamento de linguagem natural.

Por outro lado, em Banco de Dados, área na qual se encaixa este trabalho, o estudo é voltado para a criação de linguagens de consulta com preferências, capazes de interagir com bases de dados personalizadas [Kießling and Köstler 2002, Chomicki 2003, Hristidis et al. 2001, Koutrika et al. 2006]. Mais especificamente, o interesse está em desenvolver

algoritmos que retornam as tuplas mais preferidas de acordo com uma ordem de preferência [Preisinger et al. 2006, Börzsönyi et al. 2001].

Se analisarmos a evolução das consultas em Sistemas de Gerenciamento de Bancos de Dados Relacionais (SGBDRs), perceberemos um aumento no nível de exigência do usuário em relação à informação a ser recuperada. A Figura 1.1 ilustra essa ideia. O aperfeiçoamento das consultas de maneira mais adequada às necessidades do usuário evoluiu inerentemente ao aumento no volume e tipos de dados armazenados. Inicialmente, apenas consultas que recuperavam as informações com restrições na cláusula `WHERE`, as chamadas restrições *hard*, eram suficientes. Em seguida, vieram os operadores que aplicam agrupamentos e janelamentos, representados pelas cláusulas `GROUP BY` e `ORDER BY`, além das consultas por similaridade, que surgiram com a necessidade de recuperar dados complexos, como imagens, por exemplo, que não podem ser comparadas por simples operações de igualdade. Até que hoje, os esforços estão voltados para a obtenção dos dados de maneira personalizada, através, portanto, das preferências do usuário.

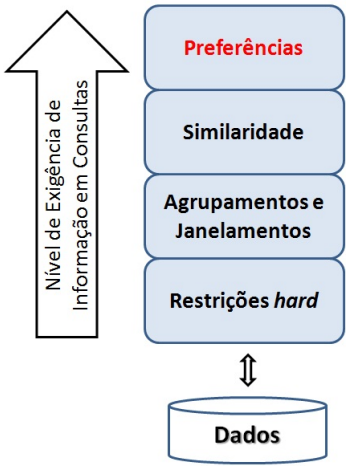


Figura 1.1: Aperfeiçoamento de consultas em SGBDRs

Nesse contexto de consultas personalizadas, em [de Amo and Ribeiro 2009] foi introduzida a linguagem CPref-SQL que permite expressar consultas contendo preferências condicionais (consultas-pc), através de uma abordagem qualitativa. A CPref-SQL é uma extensão da linguagem SQL, que incorpora, além das usuais restrições *hard* (declaradas na cláusula `WHERE`), as chamadas restrições *soft*, que são especificadas através de um conjunto de regras de preferências e referenciadas pela cláusula `ACCORDING TO PREFERENCES`. Foi proposto também o operador ***Select-Best*** para a linguagem CPref-SQL, que é responsável por obter as tuplas mais preferidas. Entretanto, não foi abordada a implementação da linguagem.

Um aspecto importante que vem sendo discutido em relação a consultas com preferências é a possibilidade do usuário poder controlar a quantidade de tuplas a serem retornadas durante suas consultas [Yiu and Mamoulis 2007, Papadias et al. 2005]. Caso

isso não aconteça, dois cenários podem existir: (1) as preferências especificadas pelo usuário podem ser muito restritivas e, assim, o conjunto das tuplas mais preferidas será muito pequeno; (2) as preferências podem ser muito permissivas, fazendo com que o usuário confronte novamente com um conjunto de dados que pode ser tão grande quanto a própria base.

Assim, é importante que uma linguagem de consultas com suporte a preferências tenha a flexibilidade de permitir que o usuário especifique a quantidade de tuplas preferidas que devem ser retornadas. Por exemplo, diante de uma lista de opções de viagens, é interessante que o usuário realize uma consulta do tipo: *"gostaria de obter uma relação contendo exatamente os 5 destinos que mais se adequam às minhas preferências."*

Nesta dissertação propomos a implementação da CPref-SQL, adicionando à linguagem o suporte a consultas top-k com preferências condicionais, ou seja, consultas que retornam as K tuplas mais preferidas pelo usuário de acordo com sua hierarquia de preferência.

1.1 Motivação

Como motivação, o Exemplo 1.1 é uma típica situação na qual consultas-pc podem ser utilizadas através da linguagem CPref-SQL.

Exemplo 1.1. Seja uma relação de um banco de dados chamada `Viagens`, que armazena informações sobre opções de pacotes turísticos de uma agência, com os atributos: `Destino` (D), `Roteiro` (R), `Preço` (em R$) (P) e `Duração` (em dias) (Du). As seguintes afirmações expressam minhas preferências sobre viagens:

- Em geral, prefiro que sejam viagens com preço inferior a R$3000, independente do valor dos demais atributos;
- Se o preço for maior que R$3000, prefiro ir à praia do que roteiros ecológicos, independente do destino e duração da viagem;
- Se o preço for maior que R$3000 e for uma opção de roteiro ecológico, então prefiro que seja uma duração de até 5 dias, para qualquer destino.

Gostaria de obter os quatro pacotes que mais se adequam às minhas preferências, desde que não sejam opções com preço acima de R$5000. A consulta-pc top-k correspondente é:

```
CREATE PREFERENCES myprefs
 FROM Viagens AS
   P < 3000 > P ≥ 3000 [D, R, Du] AND
   IF P > 3000 THEN R = 'praia' > R = 'ecologico' [D, Du] AND
   IF R = 'ecologico' AND P > 3000 THEN Du ≤ 5 > Du > 5 [D];

SELECT *
FROM Viagens
```

```
WHERE P ≤ 5000
ACCORDING TO PREFERENCES (myprefs, 4);
```

Nessa consulta, a restrição *hard* é `P ≤ 5000` e as restrições *soft* são dadas pelas regras *myprefs*. Note que o parâmetro $K = 4$ também aparece dentro da cláusula `ACCORDING TO PREFERENCES`, uma vez que ele faz parte das restrições *soft*. Em geral, se K não é especificado, as tuplas mais preferidas são retornadas.

É sabido que um operador de preferências que retorna as tuplas que não são dominadas por nenhuma outra, como é o caso do ***Select-Best*** da CPref-SQL, não aumenta o poder de expressão da SQL, uma vez que ele pode ser expresso em álgebra relacional [Chomicki 2003]. Ou seja, é possível expressar consultas em SQL que têm a mesma semântica desses operadores. Entretanto, estender a SQL com novos operadores é importante para podermos expressar preferências de maneira mais intuitiva, além do fato de que algoritmos específicos que resolvem as operações de preferência têm, potencialmente, maior performance. É por esses motivos que, neste trabalho, damos continuidade às propostas de [de Amo and Ribeiro 2009].

Para concretizarmos, então, a implementação da linguagem CPref-SQL, nossos principais objetivos são (1) estender o modelo de preferência da CPref-SQL de maneira a torná-lo mais genérico e flexível, no qual mais tipos de consultas-pc podem ser expressas; (2) incorporar à linguagem CPref-SQL o novo operador ***SelectK-Best***, responsável por avaliar consultas-pc top-k e (3) codificar a linguagem CPref-SQL através de novos algoritmos para os operadores ***Select-Best*** e ***SelectK-Best***.

Por fim, diante do avanço de sistemas que manipulam dados complexos como imagens e áudio, torna-se interessante uma integração entre aspectos de similaridade e preferências em uma consulta sobre dados convencionais e complexos. Uma integração significa a possibilidade de realizar consultas do tipo: *"Dentre as minhas viagens mais preferidas, liste aqueles destinos que mais se assemelham a essa imagem de um ponto turístico."*

Essa possibilidade de integração deu origem a um quarto objetivo em nosso trabalho, que é (4) apresentar aplicações da linguagem CPref-SQL que implementamos, no contexto de combinação entre os operadores de preferência e similaridade, para realizar consultas que incorporam as duas abordagens ao mesmo tempo.

1.2 Contribuições

As principais contribuições deste trabalho através das quais atingimos nossos objetivos, são:

1. Extensão do modelo de preferência da CPref-SQL para suporte não só ao predicado do igualdade (=), como também a predicados mais genéricos, tais como: $>$, $<$, $\geq$, ...;

2. Inserção de uma nova ordem de preferência, a ordem fraca, no modelo de preferência da CPref-SQL, tornando-o menos conservativo ao comparar tuplas;
3. Implementação do teste de consistência que garante que o conjunto de regras expressas pelo usuário não contenha inconsistências;
4. Introdução do novo operador ***SelectK-Best*** que resolve consultas-pc top-k na CPref-SQL;
5. Proposta dos algoritmos **strongDomTest** e **weakDomTest** que calculam a ordem de preferência entre duas tuplas sob as abordagens de ordem forte e ordem fraca, respectivamente;
6. Proposta dos algoritmos **G-BNL** e **GRank-BNL** que implementam os operadores ***Select-Best*** e ***SelectK-Best***, respectivamente;
7. Duas implementações da linguagem CPref-SQL feitas diretamente no processador de consultas do SGBDR PostgreSQL, que resolvem consultas-pc com os predicados $=, >, <, \geq, \leq$ e $\neq$. Uma adota ordem de preferência forte, a outra utiliza a ordem de preferência fraca;
8. Uma versão simplificada da CPref-SQL implementada de maneira independente do código-fonte do SGBDR, para comparações de desempenho;
9. Proposta de tradução de consultas CPref-SQL sob a ordem forte de preferência em SQL recursivo;
10. Testes de performance e escalabilidade que comparam as implementações realizadas;
11. Duas aplicações da CPref-SQL no contexto de consultas por similaridade: combinação dos operadores de preferência e similaridade e obtenção automática de regras-pc.

1.3 Organização da dissertação

Esta dissertação está organizada da seguinte maneira:

Capítulo 2 - Trabalhos Correlatos e Fundamentos Teóricos. São apresentados os principais trabalhos correlatos referentes aos diversos temas envolvidos nesta dissertação. Em paralelo, os conceitos necessários para o entendimento da linguagem CPref-SQL são definidos.

Capítulo 3 - A Linguagem CPref-SQL. O modelo de preferência estendido e as ordens de preferência são definidos, descrevendo a nova semântica da linguagem CPref-SQL. Também é apresentada a implementação do teste de consistência. Ao fim, o novo operador ***SelectK-Best*** e a sintaxe da linguagem são detalhados.

Capítulo 4 - Algoritmos e Implementação. São descritos os algoritmos G-BNL e GRank-BNL e os algoritmos para determinar as ordens de preferência forte e fraca. Além disso, neste capítulo são mostrados detalhes de implementação e disponibilização dos códigos. É discutida também uma maneira de traduzir consultas CPref-SQL em linguagem SQL com recursão.

Capítulo 5 - Resultados Experimentais. Expõe os testes realizados sobre a linguagem implementada, que comparam a performance e escalabilidade entre os algoritmos da CPref-SQL e as consultas traduzidas em SQL.

Capítulo 6 - Estudos de Caso. Ilustra duas possíveis aplicações da CPref-SQL no contexto de busca por similaridade. A primeira é uma combinação entre os operadores de similaridade e de preferência que pode aprimorar o resultado de consultas sobre imagens médicas. A segunda é uma integração com um sistema de similaridade que pode fornecer automaticamente (através de mineração de dados) as preferências do usuário.

Capítulo 7 - Conclusão. São apresentadas as conclusões e perspectivas para trabalhos futuros.

Referências Bibliográficas.

Apêndice A - A Linguagem CPref-SQL na Web. Apresenta o *site* que desenvolvemos com o objetivo de organizar a documentação e *download* de códigos, bem como de divulgar o trabalho realizado.

As principais contribuições deste trabalho encontram-se nos Capítulos 3, 4, 5 e 6.

Capítulo 2

Trabalhos Correlatos e Fundamentos Teóricos

Neste capítulo são destacados diversos trabalhos já realizados na área de preferências. Como a proposta de implementação da linguagem CPref-SQL envolve diferentes assuntos na literatura, discutimos os estudos e fundamentos que nos motivaram e que são mais relevantes em cada tema, contextualizando a CPref-SQL e as novas propostas deste trabalho em relação ao estado da arte.

Na Seção 2.1 apresentamos trabalhos envolvendo raciocínio e modelagem com preferências. Na Seção 2.2 discutimos alguns operadores de preferência propostos na literatura, bem como os algoritmos que os implementam. Também expomos as principais linguagens com suporte a preferências, ressaltando suas diferenças em relação à CPref-SQL. Já a Seção 2.3 é uma discussão acerca das diferentes abordagens de consultas top-K. Na Seção 2.4 apresentamos uma breve discussão sobre as formas de implementação possíveis para um módulo de preferência. Por fim, na Seção 2.5, são feitas as considerações finais sobre este capítulo.

2.1 Raciocínio e Modelagem com Preferências

O desenvolvimento de formalismos para especificação e raciocínio com preferências tem sido uma tarefa importante, principalmente no campo de Inteligência Artificial (IA) [Boutilier et al. 2004, Brafman et al. 2006, Wilson 2004]. Enquanto em algumas situações pode ser mais natural expressar preferências de maneira quantitativa, em outras, é melhor utilizar sentenças qualitativas. Ou ainda, elas podem ser incondicionais ou condicionais. Enfim, neste tema, os pesquisadores buscam responder: qual a melhor maneira para modelar o perfil de um usuário e como inferir uma ordem de preferência entre os objetos?

Dentre as principais propostas de modelagem e raciocínio de preferências que motivaram este trabalho, estão as CP-nets e TCP-nets [Boutilier et al. 2004, Brafman et al. 2006],

as preferências condicionais propostas por [Wilson 2004], além de diferentes *frameworks* voltados para representação de preferências em bancos de dados [Chomicki 2003,Koutrika and Ioannidis 2005, Agrawal and Wimmers 2000]. A seguir detalhamos cada uma delas.

2.1.1 CP-nets e TCP-nets

CP-net é um formalismo para representação de preferências qualitativas condicionais que tem sido alvo de diversas pesquisas na área de IA [Boutilier et al. 2004, Goldsmith et al. 2008, Brafman and Domshlak 2002]. Uma CP-net é um grafo dirigido sobre um conjunto de atributos onde cada nó representa um atributo e as arestas representam as dependências condicionais. Quando há uma aresta do atributo X para o atributo Y significa que as preferências sobre o atributo Y são condicionadas pelo valor do atributo X. Cada atributo X possui vinculada ao seu respectivo nó uma *tabela de preferência condicional* (TPC) que descreve as preferências sobre o atributo X levando em conta suas dependências.

As CP-nets utilizam a semântica *ceteris paribus* para inferir uma ordem de preferência. *Ceteris paribus* é uma expressão do latim que pode ser traduzida como *"todo o mais é igual."* No contexto de preferências, significa que para comparar dois objetos a partir de uma dada preferência, todos os demais atributos que não estão envolvidos na preferência devem coincidir em seus valores. Nas CP-nets as relações de ordem podem ser obtidas por transitividade. O Exemplo 2.1 a seguir ilustra os principais conceitos das CP-nets.

Exemplo 2.1. Suponha as seguintes preferências de um usuário diante de opções de pacotes turísticos:

1. Viagens de categoria (C) internacional (i) são melhores do que viagens nacionais (n);
2. Prefiro o meio de transporte (T) aéreo (a) ao rodoviário (r);
3. Para viagens internacionais aéreas prefiro roteiros (R) urbanos (u) a ecológicos (e). Caso contrário, são melhores as viagens de roteiro ecológico.

A Figura 2.1(a) exibe uma CP-net para as preferências especificadas. A TPC associada ao nó C, por exemplo, sugere que objetos com $C = i$ são melhores que objetos com $C = n$, para todo o restante igual. Logo, o objeto $o_1 = (i, a, u)$ é preferido a $o_2 = (n, a, u)$, já que $i \succ n$ e os outros atributos assumem o mesmo valor (*aereo* e *urbano*).

A Figura 2.1(b) é um *grafo de preferência* induzido pela CP-net. Uma aresta de um objeto o_j para um objeto o_k significa que o_j é preferido a o_k. Assim, dados dois objetos $o_1 = (i, a, u)$ e $o_3 = (i, a, e)$, pela TPC do atributo roteiro (R) da CP-net, é possível inferir que o_1 é preferido a o_3. Já os objetos $o_4 = (i, r, e)$ e $o_5 = (n, a, e)$ não podem ser comparados e, portanto, são indiferentes.

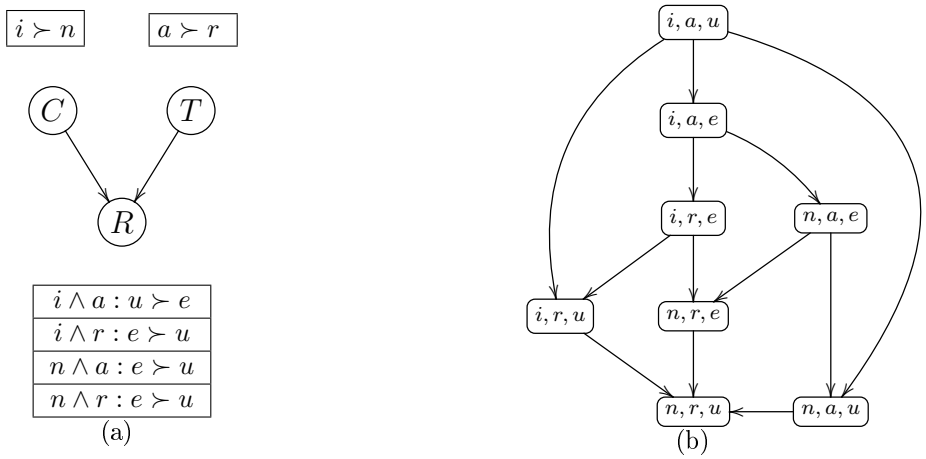


Figura 2.1: CP-net para *Viagens* e seu grafo de preferência induzido

O formalismo TCP-net (*Tradeoffs-enhanced CP-net*) é uma extensão das CP-nets que permite a representação de importância absoluta ou relativa entre atributos, e com isto, em certos casos é possível fazer comparações diretas entre objetos que diferem em mais de um atributo [Brafman and Domshlak 2002,Brafman et al. 2006]. É portanto, um modelo de preferência mais expressivo.

TCP-nets são grafos que possuem as mesmas características das CP-nets, com o adicional de mais dois tipos de arestas para representar as importâncias absoluta e relativa entre atributos. Importância absoluta é quando, dados dois atributos X e Y preferencialmente independentes, X é mais importante que Y. Isto faz com que as preferências sobre os valores de X sobreponham as preferências sobre os valores de Y. Já na importância relativa, um atributo é mais importante do que outro de acordo com os valores de um terceiro atributo ou um conjunto de atributos. Isto é, dados três atributos X, Y e Z, X é mais importante que Y dependendo do valor assumido por Z.

Exemplo 2.2. Como exemplo, suponha que o usuário dê mais importância à *categoria* de uma viagem do que ao tipo de *transporte*. Dadas as preferências do Exemplo 2.1, agora é possível dizer que $o_4 = (i, r, e)$ é preferido a $o_5 = (n, a, e)$.

2.1.2 Linguagem de Preferências Condicionais

Em [Wilson 2004] foi especificada uma linguagem de preferências condicionais que permite expressar preferências mais genéricas do aquelas usadas pelas CP-nets e TCP-nets.

A representação de preferências nessa linguagem é realizada através de *regras de preferência condicional* e *teorias de preferência condicional*, conforme as Definições 2.1 e 2.2 a seguir.

Definição 2.1. (Regra de preferência condicional) Seja $R(A_1, A_2, ..., A_n)$ um esquema relacional. Para cada atributo $A \in R$, seja dom(A) um conjunto finito de valores

de A (o domínio de A). Logo, o conjunto $\text{Tup}(R) = \text{dom}(A_1) \times \text{dom}(A_2) \times ... \times \text{dom}(A_n)$ é o conjunto de todas as tuplas possíveis de R.

Uma *regra de preferência condicional* (regra-pc) é uma sentença da forma φ: $u \rightarrow (A = a) > (A = a')[W]$, onde u é uma fórmula do tipo $(A_{i_1} = a_1) \wedge ... \wedge (A_{i_k} = a_k)$, com $A_{i_j} \in R - \{A\}$ e $a_j \in \text{dom}(A_{i_j})$ para todo $j \in \{1, ..., k\}$, a e $a' \in \text{dom}(A)$ e $W \subseteq R - \{A, A_{i_1}, ..., A_{i_k}\}$.

A fórmula u é chamada de *condição* da regra-pc φ. O conjunto de atributos que aparecem em u é denotado por *Atr(u)*.

Definição 2.2. (Teoria de preferência condicional) Uma *teoria de preferência condicional* (teoria-pc) sobre R é um conjunto finito de regras-pc.

O Exemplo 2.3 ilustra como as preferências do usuário são expressas através de uma teoria-pc.

Exemplo 2.3. Suponha que o cliente expresse as seguintes preferências sobre uma lista de viagens com os atributos `(Categoria, Transporte, Roteiro)`:

1. Para qualquer tipo de roteiro (R), prefiro viagens de categoria (C) internacional (i) a viagens nacionais (n);
2. Se a viagem for nacional, prefiro ir a um centro urbano (u) do que um roteiro ecológico (e), para qualquer meio de transporte (T).

Tais preferências podem ser representadas pela teoria-pc $\Gamma = \{\varphi_1, \varphi_2\}$, onde:

$$\Gamma = \left\{ \begin{array}{l} \varphi_1 : C = i > C = n \ [\{R\}], \\ \varphi_2 : C = n \rightarrow R = u > R = e \ [\{T\}]. \end{array} \right\}$$

Antes de descrevermos formalmente a semântica de uma regra-pc, é importante apresentar as discussões de [Wilson 2004] acerca da *consistência de uma teoria-pc.*

Um conjunto de regras-pc definido pelo usuário é passível de possuir inconsistências, podendo gerar contradições. Por exemplo, se num momento o usuário diz que prefere viagens nacionais a internacionais, pode ser que num segundo momento ele diga o contrário: viagens nacionais são melhores, fazendo com que o motor de inferência deduza que uma viagem é melhor que ela mesma!

Para evitar teorias-pc inconsistentes, foram propostas duas condições suficientes para que, dada uma teoria-pc Γ, consiga-se garantir que ela é consistente. Tais condições referem-se ao *grafo de dependência preferencial* e à *consistência local* de Γ.

Um grafo de dependência preferencial representa a relação de dependência entre os atributos que aparecem nas regras-pc de Γ. A Definição 2.3 formaliza o conceito e a Figura 2.2 ilustra o grafo de dependência preferencial da teoria-pc Γ apresentada no Exemplo 2.3.

Definição 2.3. (Grafo de dependência preferencial) Dado um conjunto de atributos A de uma teoria-pc Γ sobre A, o *grafo de dependência* sobre Γ é um grafo dirigido, definido como $G(\Gamma) = (V, E)$, onde: $V = A$ e $E = \bigcup_{\varphi_i \in \Gamma} E(\varphi_i)$, sendo $E(\varphi_i) = \{(Y, X_{\varphi_i}) \mid Y \in U_{\varphi_i}\} \cup \{(X_{\varphi_i}, Z) \mid Z \in W_{\varphi_i}\}$.

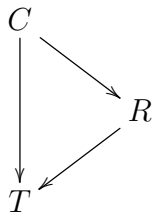


Figura 2.2: Grafo de dependência preferencial da teoria-pc Γ.

A consistência local, por sua vez, refere-se à ordem induzida sobre o domínio dos atributos. Uma teoria-pc é localmente consistente se as ordens locais produzidas em cada atributo do lado direito das regras são irreflexivas. Formalmente, tem-se:

Definição 2.4. (Consistência local) Seja Γ uma teoria-pc sobre um conjunto de atributos A. Seja $U_\Gamma = \{u_{\varphi_i} \mid \varphi_i \in \Gamma\}$. Para cada $u \in U_\Gamma$ e para cada $X \in A$, $R^*_{u,X} = \{(x_{\varphi_i}, x'_{\varphi_i}) \mid X_{\varphi_i} = X$ e u *satisfaz* $u_{\varphi_i}\}$. Seja $\succ^X_u$ o fecho transitivo sobre $R^*_{u,X}$. Uma teoria-pc Γ é *localmente consistente* se e somente se para todo $X \in A$ e para todo $u \in U_\Gamma$, a relação $\succ^X_u$ é irreflexiva.

Exemplo 2.4. Como exemplo, suponha a seguinte teoria-pc Γ':

$$\Gamma' = \left\{ \begin{array}{l} \varphi_1 : C = i > C = n, \\ \varphi_2 : R = e \rightarrow C = n > C = i. \end{array} \right\}$$

Γ' não é localmente consistente, pois a relação $\succ^C_{R=e} = \{(i, n), (n, i)\}$ é reflexiva. O par (n, i) é obtido da regra-pc φ_2, já que $R = e$ valida u_{φ_2}. O par (i, n) é obtido da regra-pc φ_1, pois u_{φ_1} é uma condição vazia e, portanto, é validada por qualquer atribuição (inclusive para $R = e$).

A garantia de consistência de uma teoria-pc é dada pelo Teorema 2.1.

Teorema 2.1. ([Wilson 2004]) *Seja uma teoria de preferência condicional* Γ*, cujo grafo de dependência preferencial é acíclico. Então,* Γ *é consistente se e somente se* Γ *é localmente consistente.*

Com isso, a teoria-pc Γ do Exemplo 2.3 é garantidamente consistente, pois seu grafo é acíclico e suas regras localmente consistentes.

Uma vez garantida a consistência de uma teoria-pc, descrevemos a seguir a semântica que um conjunto de regras-pc impõe.

Uma teoria-pc Γ sobre R induz uma *ordem de preferência* sobre $Tup(R)$. A Definição 2.5 descreve a chamada ordem de preferência induzida sobre regras de preferências condicionais.

Definição 2.5. (Ordem de preferência condicional) Seja uma regra-pc $\varphi : u \rightarrow (A = a) > (A = a')[W] \in \Gamma$. Seja $\mathbf{t} = (\mathbf{b}, \mathbf{c}, a, \mathbf{w})$ e $\mathbf{t'} = (\mathbf{b}, \mathbf{c}, a', \mathbf{w'}) \in \text{Tup}(R)$, onde $\mathbf{b}$ é uma tupla sobre *Atr(u)* que satisfaz a fórmula u, $\mathbf{c}$ é uma tupla sobre $R - (Atr(u) \cup W \cup \{A\})$, $\mathbf{w}$ e $\mathbf{w'}$ são tuplas sobre W. Diz-se que $\mathbf{t}$ é *preferido* a $\mathbf{t'}$ de acordo com φ.

O conjunto de pares de tuplas $(\mathbf{t}, \mathbf{t'})$ onde $\mathbf{t}$ é preferido a $\mathbf{t'}$ de acordo com φ é denotado por $>_\varphi$. Denota-se $>_\Gamma$ o *fecho transitivo* da relação binária $\bigcup_{\varphi \in \Gamma} >_\varphi$.

Exemplo 2.5. Como exemplo, suponha as tuplas $t_1 = (i, a, e)$, $t_2 = (n, a, u)$ e $t_3 = (n, r, e)$. De acordo com Γ do Exemplo 2.3, pode-se inferir que $t_1 >_{\varphi_1} t_2$, pois t_1 *é preferido* a t_2 pela regra φ_1. E, $t_2 = (n, a, u)$ *é preferido* a $t_3 = (n, r, e)$ pela regra φ_2. Por transitividade, $t_1 >_\Gamma t_3$.

Veja que as preferências de Γ do Exemplo 2.3 não podem ser expressas pela abordagem de CP-nets. Elas são mais relaxadas na semântica *ceteris paribus* devido os conjuntos $\{R\}$ e $\{T\}$ das regras φ_1 e φ_2, respectivamente. Entretanto, toda CP-net é uma teoria de preferência condicional. As preferências do Exemplo 2.1 de CP-nets podem facilmente ser traduzidas em uma teoria-pc $\Gamma'' = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5, \varphi_6\}$, bastando considerar $W = \emptyset$ para cada regra-pc (vide Definição 2.1):

$$\Gamma'' = \left\{ \begin{array}{l} \varphi_1 : C = i > C = n, \\ \varphi_2 : T = a > T = r, \\ \varphi_3 : C = i \wedge T = a \rightarrow R = u > R = e, \\ \varphi_4 : C = i \wedge T = e \rightarrow R = e > R = u, \\ \varphi_5 : C = n \wedge T = a \rightarrow R = e > R = u, \\ \varphi_6 : C = n \wedge T = e \rightarrow R = e > R = u. \end{array} \right\}$$

Por fim, destacamos ainda, uma segunda definição acerca da relação de ordem de uma teoria-pc, apresentada em [Wilson 2004], e denominada *ordem de preferência lexicográfica condicional parcial* (ordem lcp). É uma segunda maneira para comparar duas tuplas de acordo com uma teoria-pc, mais genérica que a ordem de preferência da Definição 2.5, que motivou diretamente o formalismo da linguagem CPref-SQL proposto neste trabalho, conforme discutiremos no Capítulo 3. A Definição 2.6 descreve uma ordem lcp.

Definição 2.6. (Ordem lexicográfica condicional parcial) Seja Γ uma teoria-pc localmente consistente sobre um conjunto V de atributos, tal que $G(\Gamma)$ é acíclico. Seja α, $\beta \in \text{Tup}(V)$. Define-se $\Delta(\alpha, \beta)$ o conjunto de atributos em V onde α e β se diferem, ou seja, $\{X \in V \mid \alpha[X] \neq \beta[X]\}$.

Para um subconjunto W de nós de G e considerando G^o o fecho transitivo de G, define-se $min_{G^o}(W)$ o conjunto $\{X \in W \mid \forall Y \in W, (Y, X) \notin G^o\}$, ou seja, o conjunto de atributos X de W que não têm ancestrais em W, de acordo com G.

Uma tupla α é dita preferida a β de acordo com Γ (denotado por $\alpha \succ_\Gamma \beta$) se para todo $X \in min_{G(\Gamma)^o}(\Delta(\alpha, \beta))$ tem-se $(\alpha[X], \beta[X]) \in \succ_\alpha^X \cap \succ_\beta^X$.

Exemplo 2.6. Como exemplo, suponha que queiramos inferir uma ordem lcp entre as tuplas $t_1 = (i, r, e)$ e $t_2 = (n, a, u)$, de acordo com a teoria-pc Γ do Exemplo 2.3. Então, temos: $\Delta(t_1, t_2) = \{C, T, R\}$ e $min_{G(\Gamma)^o}(\Delta(t_1, t_2)) = \{C\}$. Como o par $(i, n) \in \succ_{t_1}^{C} \cap \succ_{t_2}^{C}$, inferimos que $t_1 \succ_\Gamma t_2$. Note que t_1 e t_2 são incomparáveis segundo a ordem de preferência da Definição 2.5.

O Teorema 2.2, enunciado por [Wilson 2004], estabelece uma relação entre as ordens de preferência existentes em sua linguagem de preferências condicionais.

Teorema 2.2. ([Wilson 2004]) *A ordem de preferência condicional está contida na ordem lcp.*

2.1.3 *Frameworks* em Banco de Dados

É importante mencionarmos que foi a partir dos formalismos estudados em IA que surgiram vários *frameworks* voltados para representação e manipulação de preferências em bases de dados.

Em [Koutrika et al. 2011] é apresentada uma pesquisa que discute acerca dos desafios da comunidade de banco de dados em trabalhar com o tópico de preferências. Os autores propõem um *framework* para estudar diferentes abordagens que lidam com preferências em bancos de dados. Basicamente, as abordagens podem ser divididas em três categorias principais: representação, combinação e aplicação de preferências em processamento de consultas.

Alguns exemplos de *frameworks* de preferências em bancos de dados são os trabalhos [Agrawal and Wimmers 2000,Koutrika and Ioannidis 2005,Chomicki 2003]. Em [Agrawal and Wimmers 2000], as preferências são expressas através de uma função de *score* e, portanto, tratadas de maneira quantitativa. Já em [Koutrika and Ioannidis 2005] o usuário associa um grau de interesse a possíveis valores da base de dados e as preferências são modeladas através de uma rede de preferências (grafo direcionado acíclico). No trabalho de [Chomicki 2003] as preferências são expressas através de fórmulas da lógica de primeira ordem sobre os atributos de uma relação, denominadas *fórmulas de preferência*.

Em especial, o formalismo da CPref-SQL em [de Amo and Ribeiro 2009,Ribeiro 2008] é todo fundamentado no trabalho de [Wilson 2004], descrito anteriormente. As preferências são expressas por teorias-pc que induzem uma ordem sobre as tuplas do banco de dados. Neste trabalho, propomos um extensão desse modelo de preferência da CPref-SQL originalmente proposto, com o objetivo de aumentar seu poder de expressão. Maiores detalhes acerca do modelo da linguagem e as nossas contribuições sobre esse modelo serão discutidos no Capítulo 3.

2.2 Operadores, Algoritmos e Extensões SQL para Avaliação de Preferências

Quando um usuário expressa suas preferências sobre um conjunto de elementos, mesmo que implicitamente, existe uma relação binária de ordem imposta sobre os elementos. Além do mais, assim como é possível realizar operações sobre conjuntos (união, interseção, etc), existem também as operações sobre preferências. E é essa a principal função dos diferentes operadores propostos na literatura [Chomicki 2003,Börzsönyi et al. 2001]: qual a melhor forma de lidar com as relações e operações de preferência? Ou ainda, qual a melhor semântica para escolha das preferidas?

Neste tópico, selecionamos as principais propostas de operadores de preferência e algoritmos que os implementam. Vale ressaltar que não temos como objetivo discutir acerca dos fundamentos teóricos em manipulação de preferências como relações de ordem. Maiores detalhes podem ser encontrados nos estudos descritos em [Ribeiro 2008].

Como consequência direta dos novos operadores, na área de banco de dados, o desenvolvimento de linguagens de consulta com suporte a preferências tem sido abordado em diferentes trabalhos. Como exemplo, podemos citar desde a extensão da linguagem SQLf para preferências baseadas na lógica *fuzzy* [Ludovic et al. 2007], até consultas em web semântica com extensão da SPARQL [Siberski et al. 2006].

A seguir, destacamos as principais propostas correlatas à linguagem CPref-SQL neste tema: o operador *Skyline* [Börzsönyi et al. 2001], o operador *winnow* [Chomicki 2003] e a linguagem Preference SQL [Kießling and Köstler 2002].

2.2.1 O Operador *Skyline*

Em [Börzsönyi et al. 2001] foi proposta uma operação que estende os sistemas de banco de dados, chamada *Skyline*. Dado um grande conjunto de pontos, essa operação filtra os pontos interessantes. Um ponto é interessante se ele não é dominado por nenhum outro ponto. Por exemplo, uma viagem pode ser interessante para um turista se nenhuma outra opção é mais barata e tem maior duração. Um ponto p_1 domina outro ponto p_2 se duas condições forem satisfeitas ao mesmo tempo:

1. p_1 é igual ou melhor do que p_2 em todas as dimensões e
2. p_1 é melhor que p_2 em pelo menos uma dimensão.

A Figura 2.3 mostra o conjunto de pontos interessantes a partir de uma operação *Skyline* que busca as viagens de maior duração e mais baratas. É possível perceber, por exemplo, que uma viagem com preço de R$1500 e duração de 10 dias *domina* uma viagem que custa R$2000 e dura apenas 6 dias.

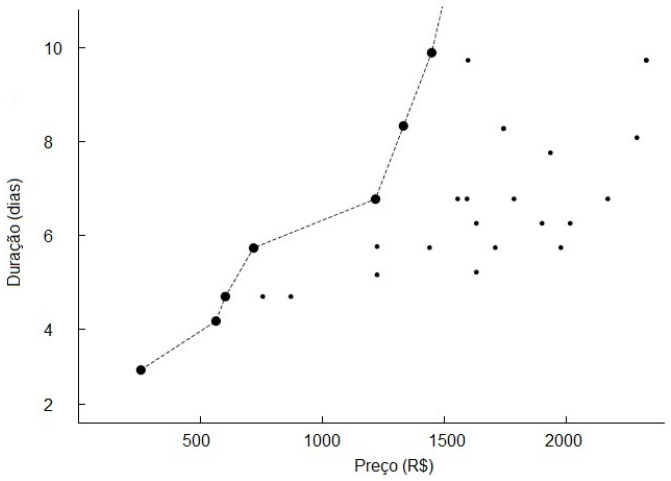


Figura 2.3: *Skyline* de Viagens

Dado o operador *Skyline*, a proposta é estender a SQL com a cláusula `SKYLINE OF`. A consulta correspondente à preferência por viagens com menor preço e maior duração pode ser feita da seguinte maneira:

```
SELECT * FROM Viagens
SKYLINE OF preco MIN, duracao MAX;
```

Para calcular o conjunto dos pontos *Skyline*, em [Börzsönyi et al. 2001] também são propostos diversos algoritmos. Dentre eles, algoritmos BNL (*block-nested-loop*), algoritmos *divide and conquer* (dividir e conquistar), além de propostas simples como a tradução de consultas *Skyline* em SQL padrão.

O algoritmo BNL básico, assim como o algoritmo BNL para operação de junção, lê repetidamente um conjunto de tuplas. Por outro lado, o algoritmo *divide and conquer* trivial, utiliza a técnica de dividir o problema em pequenos subproblemas, obtendo a cada passo, um conjunto de pontos interessantes. Já a abordagem de traduzir consultas para a SQL padrão ilustra que o *Skyline* não acrescenta poder de expressão à álgebra relacional. Entretanto, é mostrado que essa abordagem possui um desempenho menor do que as demais.

Por terem um melhor desempenho, os algoritmos BNL desse trabalho correlato serviram de base para outras propostas na literatura, como por exemplo os algoritmos BNL^{+} de [Chan et al. 2005a] e BNL^{++} de [Preisinger et al. 2006], além do BBS^{+}, SDC e SDC^{+} de [Chan et al. 2005b].

A CPref-SQL também segue as linhas do BNL de [Börzsönyi et al. 2001] em seus algoritmos.

Em [de Amo and Ribeiro 2009] foi proposto o BNL* para avaliar o operador ***Select-Best*** da linguagem. Ele utiliza um grafo *better-than* reduzido, através do qual é possível comparar duas tuplas conforme suas listas de escopos e decidir se há uma ordem de

preferência entre elas. Esse algoritmo, entretanto, não foi implementado.

Em [Pereira and de Amo 2010a] propusemos e implementamos os algoritmos BNL** e R-BNL** (*Ranked*-BNL**). Eles avaliam o ***Select-Best*** e o novo operador ***SelectK-Best*** da CPref-SQL. Verificamos que a implementação do BNL* através do grafo reduzido é muito onerosa, daí o surgimento desses novos algoritmos. A diferença está na forma como eles comparam duas tuplas: agora, através de um Programa Datalog.

Nesta dissertação, descrevemos a implementação da linguagem através dos algoritmos G-BNL e o GRank-BNL. São novos algoritmos que também seguem as linhas do BNL de [Börzsönyi et al. 2001] para avaliar os operadores da CPref-SQL e também utilizam Programas Datalog para comparar tuplas. A novidade é que eles foram projetados para o novo modelo de preferência da linguagem, agora estendido, mais genérico e completo, conforme apresentaremos nos Capítulos 3 e 4.

2.2.2 O Operador *Winnow*

Em [Chomicki 2003] foi proposto um novo operador da álgebra relacional chamado *winnow*. Ele seleciona as tuplas mais preferidas de uma base de dados de acordo com uma dada relação de preferência.

Neste trabalho correlato, uma relação de preferência é expressa através de fórmulas da lógica de primeira ordem, denominadas *fórmulas de preferência*. Como exemplo, considere uma instância da relação `Viagens(Destino,Roteiro,Preço,Duração)`, ilustrada na Figura 2.4.

DESTINO	ROTEIRO	PREÇO	DURAÇÃO
Rio de Janeiro	praia	1500	4
Búzios	praia	3000	3
Búzios	cruzeiro	5000	7
Rio de Janeiro	urbano	5000	10
Parati	historico	2000	5

Figura 2.4: Uma instância da relação `Viagens`

Uma preferência pode ser especificada por uma fórmula de preferência C_1 da seguinte maneira:

$$(d, r, p, du) \succ_{C_1} (d', r', p', du') \equiv d = d' \wedge p < p'$$

Esta fórmula indica que entre duas opções de viagens para o mesmo destino ($d = d'$), o usuário prefere aquela com o preço menor ($p < p'$).

Assim, dada uma relação de preferência $\succ_C$ sobre uma relação r, o *winnow* é definido como:

$$w_C(r) = \{t \in r \mid \nexists t' \in r, t' >_C t\}.$$

Ou seja, o operador retorna as tuplas que não são dominadas por nenhuma outra. A

Figura 2.5 ilustra o resultado da operação $w_{C_1}(Viagens)$.

DESTINO	ROTEIRO	PREÇO	DURAÇÃO
Rio de Janeiro	praia	1500	4
Búzios	praia	3000	3
Parati	historico	2000	5

Figura 2.5: Resultado da operação $w_{C_1}(Viagens)$.

Assim como o *Skyline*, o *winnow* não aumenta o poder de expressão da álgebra relacional. Entretanto, é mostrado que a definição de um novo operador pode trazer benefícios como o estudo de técnicas de avaliação e otimização dedicadas à seleção dos melhores.

A CPref-SQL relaciona-se com esse trabalho pois seu operador ***Select-Best*** segue a mesma semântica do *winnow* para a escolha das preferidas, a chamada semântica BMO – *Best Matches Only*, inicialmente proposta em [Kießling 2002] e também utilizada na linguagem Preference SQL que descrevemos a seguir.

2.2.3 Preference SQL

A linguagem de consulta Preference SQL é uma extensão da linguagem SQL padrão que permite realizar consultas contendo preferências [Kießling and Köstler 2002]. Nessa linguagem, cada preferência é modelada como uma relação binária de ordem fraca (irreflexiva, transitiva e negativamente transitiva) [Ribeiro 2008, Chomicki 2003].

A proposta é que as preferências sejam especificadas através de um conjunto de operadores definidos por uma *álgebra de preferências* [Kießling 2002]. No modelo de preferência da Preference SQL, existem dois tipos de operadores: *built-in* e operadores complexos. Os primeiros estabelecem preferências locais sobre os *valores dos atributos*. São eles: AROUND e BETWEEN (aproximação), LOWEST e HIGHEST (mini e maximização) e POS e NEG (favoritos e não desejados). Já os operadores complexos são responsáveis pela composição de duas ou mais preferências sobre as *tuplas*, chamados CASCADE para *composição de priorização* (ordem de importância sobre os atributos) e AND para *composição Pareto* (todos os atributos têm importância igual).

O Exemplo 2.7 ilustra como é resolvida uma consulta Preference SQL com composição Pareto.

Exemplo 2.7. Seja a instância da relação `Viagens` da Figura 2.4 e a seguinte consulta Preference SQL:

```
SELECT *
FROM Viagens
PREFERRING duracao AROUND 4
   AND POS(roteiro,{praia});
```

As preferências são especificadas utilizando a cláusula `PREFERRING`. Nesta consulta há duas preferências combinadas através de uma composição Pareto (indicada pelo conectivo `AND`). A primeira preferência diz que viagens com duração por volta de 4 dias são melhores. A segunda expressa que o roteiro praia é favorito.

O cálculo das tuplas preferidas é feito através de *funções de ranking* para cada atributo. Para o atributo duração, a função é definida da seguinte maneira: $distance(du) = n$ se $|valor(du) - valorotimal| = n$. O objetivo é minimizar $distance(du)$. De acordo com a primeira preferência, as melhores tuplas são aquelas com duração próxima a 4 (valor otimal). Já para o roteiro, a função é a seguinte: $level(r) = 1$ se $valor(r) = praia$. Caso contrário, $level(r) = 2$. Logo, as melhores tuplas são aquelas com $level(r) = 1$. A Figura 2.6 ilustra as tuplas com seus respectivos *rankings*.

	DESTINO	ROTEIRO	PREÇO	DURAÇÃO	distance(du)	level(r)
t_1	Rio de Janeiro	praia	1500	4	0	1
t_2	Búzios	praia	3000	3	1	1
t_3	Búzios	cruzeiro	5000	7	3	2
t_4	Rio de Janeiro	urbano	5000	10	6	2
t_5	Parati	historico	2000	5	1	2

Figura 2.6: Funções de ranking da instância de `Viagens`

De acordo com [Kießling 2002], a composição Pareto $>_{P_{1,2}} = >_{P_1} \otimes >_{P_2}$ é definida da seguinte maneira: $(a, b) >_{P_{1,2}} (a', b')$ se e somente se $a \geq_{P_1} a' \wedge b \geq_{P_2} b' \wedge (a >_{P_1} a' \vee b >_{P_2} b')$. Logo, nesse exemplo, seguindo a semântica BMO, as tuplas preferidas são t_1 e t_2.

É importante mencionar que o resultado da acumulação Pareto de ordens fracas sobre o domínio dos atributos é uma ordem parcial estrita sobre o conjunto de tuplas [Chomicki 2003]. Por serem irreflexivas, preferências do tipo "prefiro X a X" nunca serão inferidas. Logo, nessa abordagem, questões de consistência não precisam ser levadas em conta, diferentemente das preferências condicionais de [Wilson 2004], discutidas na seção anterior, e que são a base da CPref-SQL.

As consultas Pareto (forma como ficaram conhecidas as consultas em Preference SQL) são mais genéricas que as consultas *skyline*. Em [Kießling 2002] também é mostrado que a cláusula `SKYLINE` é um subconjunto da Preference SQL.

É, portanto, um exemplo de linguagem de extensão da SQL com suporte a preferências, que adota modelos e definições ortogonais à CPref-SQL.

2.3 Consultas Top-K

Conforme indicado por diversos autores [Yiu and Mamoulis 2007,Hristidis et al. 2001, Papadias et al. 2005], é interessante que o tamanho do conjunto de tuplas mais preferidas a ser retornado possa ser controlado pelo usuário, pois mesmo com restrições *soft*, pode ser

que as preferências sejam muito restritivas e retornem poucas tuplas, ou muito relaxadas, retornando quase que a base inteira novamente.

O tópico de introdução do controle do usuário para delimitar o tamanho do conjunto de respostas foi tratado num trabalho pioneiro de [Carey and Kossman 1997], através das consultas *Top N*. Uma consulta *Top N* retorna as N primeiras tuplas de acordo com a ordem que aparecem no banco de dados. Para indicar uma consulta *Top N* foi proposta a nova cláusula `STOP AFTER`, uma extensão da SQL.

No contexto de preferências, diversas propostas lidam com o desafio de elencar exatamente as K tuplas de acordo com uma hierarquia de preferências do usuário. Em [Hristidis et al. 2001], por exemplo, as consultas top-K foram introduzidas numa ferramenta de modelagem de preferências quantitativas, denominada PREFER. O usuário informa a importância relativa dos atributos e também a quantidade K de tuplas que deseja como resposta. O objetivo é retornar as K tuplas com os maiores *rankings* obtidos a partir das escolhas do usuário.

Já as consultas top-K dominantes foram introduzidas em [Papadias et al. 2005] como uma extensão das consultas *Skyline*. Nessa abordagem, uma consulta top-K dominante retorna as K tuplas que dominam a maior quantidade de tuplas na base de dados. Esse conceito é ortogonal às consultas-pc com o operador ***SelectK-Best*** da CPref-SQL que propomos neste trabalho. Em [Yiu and Mamoulis 2009] uma série de algoritmos é introduzida para avaliar as consultas top-K dominantes.

Por fim, vale ressaltar a proposta do trabalho de [Lin et al. 2007]. Nessa abordagem, também voltada para consultas *Skyline*, a ideia é obter a partir das tuplas preferidas, as K tuplas que dominam a maior quantidade de outras tuplas. Portanto, é uma combinação entre a seleção dos pontos mais interessantes e, dentre eles, os top-K dominantes.

A Figura 2.7 ilustra os critérios para obtenção das top-K tuplas, de acordo com as diferentes abordagens apresentadas. Nas consultas *Skyline*, o contexto é a escolha de viagens com o menor preço e menor distância da cidade em que o usuário se encontra, ou seja, as melhores opções são aquelas com preço e distância minimizados.

Veremos no Capítulo 3 que a proposta do nosso trabalho é uma nova abordagem, diferente dessas apontadas nos trabalhos correlatos. Para nós, a escolha dos K melhores depende do *nível* que as tuplas ocupam a partir de uma hierarquia de preferências.

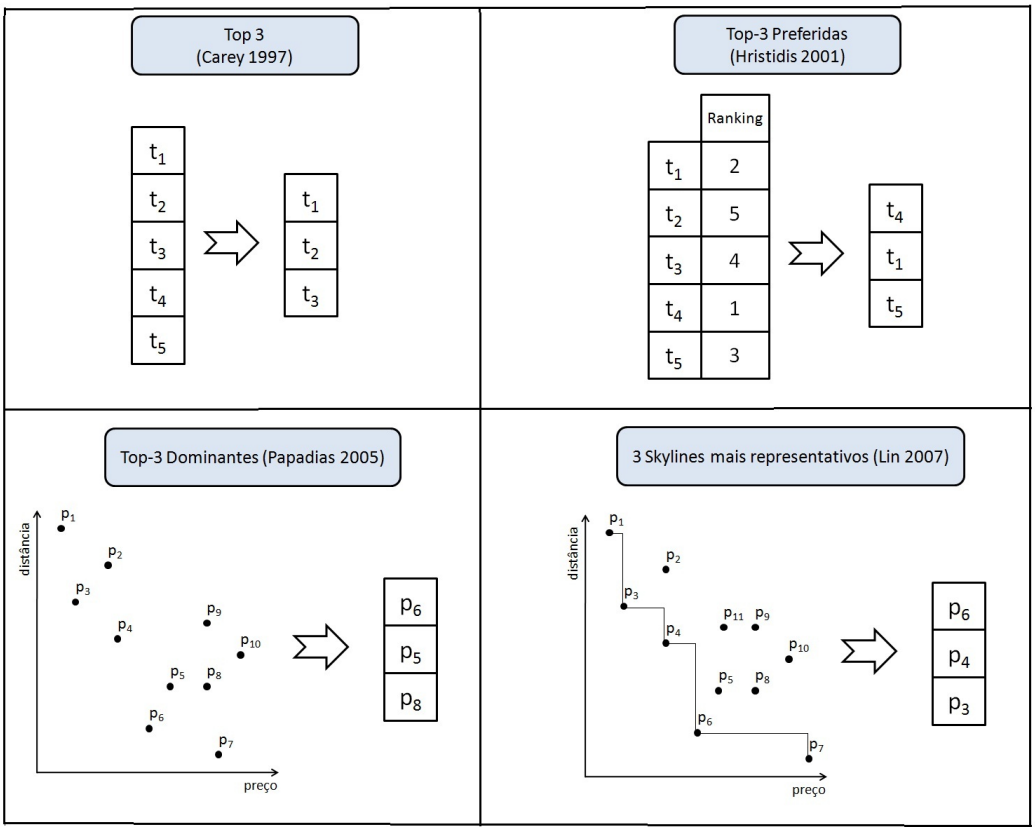


Figura 2.7: Abordagens de consultas top-K

2.4 Implementação de Módulos de Preferência

Na literatura, diversos modelos de preferência foram implementados. Conforme discutimos nas seções anteriores, top-k, *Skylines*, consultas Pareto e, principalmente, preferências condicionais são alguns exemplos. Uma questão fundamental subjacente a cada modelo é a técnica utilizada para implementá-lo.

Em [Levandoski et al. 2010b] é feita uma análise sobre as possíveis formas de implementação de módulos de preferência, descritas a seguir.

1. **Tradução em SQL:** essa abordagem é a mais simples, mas quase sempre com menor performance. Segundo mostramos na Seção 2.2, a maioria dos modelos de preferência não aumentam o poder de expressão da álgebra relacional e, portanto, é possível expressar uma consulta com preferências apenas com a linguagem SQL padrão e submetê-la ao SGBDR. Exemplos de experimentos realizados com essa abordagem são o operador *Skyline* no trabalho de [Börzsönyi et al. 2001] e a Preference SQL de [Kießling and Köstler 2002].

2. ***On-top*:** é a abordagem mais comum, na qual o método de preferência é implementado como um programa *stand-alone* ou através de funções definidas pelo usuário no SGBDR. Assim, o SGBDR é tratado como uma caixa-preta e os operadores de preferência são totalmente desacoplados das operações internas do banco (junções e

seleções, por exemplo). A vantagem é a simplicidade e flexibilidade, pois com uma implementação *on-top* é possível manipular preferências sobre diferentes SGBDRs, em suas diferentes versões. Como exemplo existem diversos trabalhos que adotaram essa abordagem: [Chan et al. 2005a, Chan et al. 2005b, Preisinger et al. 2006, Preisinger and Kießling 2007, Lin et al. 2007, Yiu and Mamoulis 2009, Yiu and Mamoulis 2007].

3. ***Built-in*:** método mais eficiente para avaliação de preferências em banco de dados. Aqui, os operadores de preferência são acoplados diretamente no processador de consultas do SGBDR, possibilitando a criação de operações otimizadas, customizadas e que interagem com as operações internas já existentes. A desvantagem desse método é que a linguagem fica disponível apenas para o SGBDR específico no qual foi implementado, mas com certeza é o mais eficiente. Os trabalhos [Eder 2009, Eder and Wei 2009] são exemplos de implementações do operador *Skyline* diretas no código-fonte do PostgreSQL.

Dadas essas três abordagens, em [Levandoski et al. 2010b] os autores propõem o *FlexPref*, um *framework* extensível para avaliação de preferências em bancos de dados. A proposta do *FlexPref* é já fornecer todo o motor de busca de uma consulta com preferência, independente do método. Ele é flexível quanto ao modelo de preferência, pois pode implementar um novo método de preferência apenas com a definição, por parte do usuário, de três funções básicas, que capturam a essência do novo modelo. A Figura 2.8 ilustra a arquitetura do framework proposto.

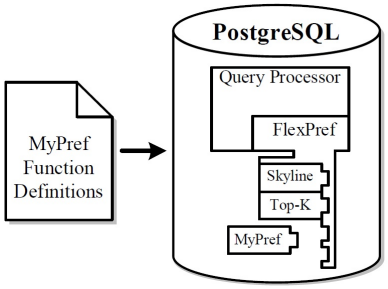


Figura 2.8: Arquitetura FlexPref ([Levandoski et al. 2010b])

É, portanto, uma abordagem *centrista* que combina os métodos *on-top* e *built-in*. *On-top* porque integrar um novo modelo de preferência significa apenas registrar funções, e *built-in* porque o motor de busca está implementado diretamente no SGBDR PostgreSQL. Os trabalhos [Levandoski et al. 2010a, Levandoski et al. 2010c] estendem essa proposta.

Diante de diferentes formas de implementação e possibilidades, a implementação da CPref-SQL, neste trabalho, é focada na abordagem *built-in*. Entretanto, mostramos também como traduzir algumas consultas CPref-SQL em SQL com recursão, além de um protótipo de implementação *on-top*. No Capítulo 5 comparamos as diferentes abordagens implementadas através de experimentos.

2.5 Considerações Finais

Neste capítulo foram apresentados os principais trabalhos sobre o tratamento de preferências. Mostramos que muitas propostas de linguagens ou operadores de preferência em bancos de dados são fundamentadas em formalismos para modelagem e raciocínio com preferências, estudados no campo de inteligência artificial. Além disso, exemplificamos o estado da arte do tópico de consultas top-K com preferências, através de trabalhos que tratam o tema sob diferentes abordagens. Por fim, tratamos a questão de implementação de uma linguagem de preferências, discutindo as características das possibilidades existentes.

Capítulo 3

A Linguagem CPref-SQL

A proposta de uma linguagem de consulta que estende a linguagem SQL para suporte a preferências envolve desde a definição de um modelo de preferência, até a proposição de operadores algébricos, algoritmos e, de fato, sua implementação.

Conforme mostramos no Capítulo 2, este trabalho é uma continuação do trabalho iniciado em [de Amo and Ribeiro 2009], no qual a CPref-SQL foi proposta. Mais especificamente, a CPref-SQL foi concretizada em tal trabalho correlato com as seguintes características: introdução da noção de consultas com preferências condicionais (consultas-pc), que seguem o formalismo de [Wilson 2004] como modelo de preferência; extensão da álgebra relacional da SQL com o novo operador ***Select-Best***, que obtém as tuplas dominantes, e algoritmo BNL* que avalia o operador ***Select-Best***. Entretanto, não foi abordada a implementação da linguagem.

Neste trabalho, aprimoramos a CPref-SQL. Estendemos seu modelo de preferência para suporte a regras com predicados genéricos, com dois tipos de ordem de preferência, com o intuito de torná-lo mais expressivo e completo; introduzimos o conceito de consultas top-K na linguagem, através do novo operador ***SelectK-Best*** e propusemos e implementamos novos algoritmos G-BNL e GRank-BNL, referentes aos operadores ***Select-Best*** e ***SelectK-Best***, respectivamente.

A Tabela 3.1 sintetiza as novas propostas deste trabalho em relação à proposta inicial da linguagem CPref-SQL de [de Amo and Ribeiro 2009].

	[de Amo and Ribeiro 2009]	**Nesta dissertação**
Modelo	Consultas-pc ordem forte predicado =	Consultas-pc ordem forte e ordem fraca predicados genéricos
Operadores	Select-Best	Select-Best e SelectK-Best
Algoritmos	BNL*	G-BNL e GRank-BNL
Implementação	//	Abordagem *built-in*

Tabela 3.1: Características da linguagem CPref-SQL

Neste capítulo, então, detalhamos nossas principais contribuições. Na Seção 3.1 apresentamos o novo modelo de preferência proposto e, na Seção 3.2, expomos os operadores da linguagem, a sintaxe e o plano de consulta que é executado quando uma consulta é submetida. Finalmente, na Seção 3.3 são feitas as considerações finais do capítulo.

3.1 O Modelo de Preferência

Modelo de preferência é um formalismo lógico para especificação e raciocínio com preferências. Neste trabalho, o modelo de preferência da linguagem CPref-SQL segue as linhas do formalismo de [Wilson 2004], assim como em [de Amo and Ribeiro 2009], no sentido de representar os desejos do usuário através de regras de preferências condicionais (regras-pc) e teorias de preferências condicionais (teorias-pc), que, por sua vez, induzem uma ordem de preferência sobre as tuplas de uma relação (vide Subseção 2.1.2 do capítulo anterior).

Entretanto, propomos uma extensão do modelo original: o suporte a predicados genéricos nas preferências. Como predicados genéricos, definimos aqueles que são diferentes da simples seleção por igualdade nos átomos compondo as regras de preferência. Por exemplo, os demais predicados *built-in*: $>$, $<$, $\geq$, $\leq$, $\neq$ e predicados de similaridade.

A motivação para estendermos o modelo inicial é a possibilidade de aumentar o poder de expressão das regras-pc da CPref-SQL e, dessa maneira, poder expressar preferências do tipo: *"dentre os pacotes turísticos com preço* ***menor*** *que R$2000, prefiro as opções de cruzeiro;"* ou ainda, *"prefiro viagens com duração* ***maior*** *do que quatro dias."*

A seguir, formalizamos o modelo de preferência estendido da CPref-SQL.

3.1.1 Teoria de Preferência Condicional

Definição 3.1. (Regra de Preferência Condicional) Uma regra de preferência condicional (regra-pc) sobre um conjunto de atributos V é uma expressão φ do tipo:

$$\varphi : u \rightarrow Q_1(X) > Q_2(X)[W]$$

onde:

- $X \in V$, $W \subseteq V$, $X \notin W$;
- $Q_i(X)$ (para $i = 1, 2$) é um predicado unário sobre $\mathbf{dom}(X)$ (o domínio de X);
- $\{x \in \mathbf{dom}(X) \mid x \models Q_1(X)\} \cap \{x \in \mathbf{dom}(X) \mid x \models Q_2(X)\} = \emptyset$;
- u é chamado de *condição* da regra-pc φ. É uma conjunção simples de expressões do tipo: $P_1(A_1) \wedge P_2(A_2) \wedge ... \wedge P_k(A_k)$, onde cada $P_i(A_i)$ é um predicado unário sobre o atributo A_i (para $i = 1, 2, ..., k$);

- Seja Atr(u) o conjunto de atributos que aparecem em u. Então, $(\{X\} \cup W) \cap \text{Atr}(u) = \emptyset$.

Uma tupla u sobre V é dita *compatível* com uma regra-pc φ se para cada atributo A que aparece do lado esquerdo de φ, temos que $u[A]$ satisfaz $P_\varphi(A)$, onde $u[A]$ denota o valor do atributo A na tupla u e $P_\varphi(A)$ denota o predicado P associado ao atributo A na regra-pc φ.

Assim como na Definição 2.2, a representação de um conjunto maior de preferências é feita através de uma *teoria de preferência condicional*, dada pela Definição 3.2.

Definição 3.2. (Teoria de preferência condicional) Uma teoria de preferência condicional (teoria-pc) sobre V é um conjunto finito de regras-pc sobre V.

O Exemplo 3.1 ilustra como os desejos do usuário são expressos através de uma teoria-pc, num contexto de preferências sobre opções de viagens.

Exemplo 3.1. Seja uma relação de um banco de dados chamada `Viagens`, que armazena informações sobre opções de pacotes turísticos de uma agência, com os atributos: `Destino` (D), `Roteiro` (R), `Preço` (em R$) (P), `Duração` (Du) e `Categoria` (C). A Figura 3.1 ilustra uma instância dessa relação.

	DESTINO	ROTEIRO	PREÇO	DURAÇÃO	CATEGORIA
t_1	Bonito	ecologico	2000	5	nacional
t_2	Buzios	cruzeiro	2500	7	nacional
t_3	Honolulu	praia	3000	5	internacional
t_4	Curitiba	urbano	4000	10	nacional
t_5	Nova York	urbano	4000	3	internacional
t_6	Rio de Janeiro	urbano	2500	4	nacional

Figura 3.1: Uma instância da relação `Viagens`

Suponha que a teoria-pc $\Gamma = \{\varphi_1, \varphi_2, \varphi_3\}$ expresse as preferências do usuário em relação às melhores opções de viagens:

$$\varphi_1 : (P < 3000) > (P \geq 3000)\ [\{D, R, Du\}];$$
$$\varphi_2 : (R = urbano) > (R = cruzeiro)\ [\{D, Du, C\}];$$
$$\varphi_3 : (R = urbano) \rightarrow (C = nacional) > (C = internacional)\ [\{D, Du\}].$$

Para a terceira regra φ_3, por exemplo, temos o predicado $(R = urbano)$ compondo a condição da regra, do lado esquerdo, e $(C = nacional)$ e $(C = internacional)$ do lado direito, são os consequentes. A tupla t_1 não é compatível com a regra-pc φ_3, pois $t_1[R] \neq urbano$, mas é compatível com a regra-pc φ_1, pois o conjunto de antecedentes desta é vazio.

Afinal, o que significa exatamente uma regra-pc como a φ_1 do Exemplo 3.1? Como é possível utilizarmos a teoria-pc Γ para a escolha das tuplas preferidas? De fato, a semântica de uma regra-pc está relacionada com a *ordem de preferência* induzida pela teoria-pc sobre as tuplas da relação.

3.1.2 Ordem de Preferência

Seja $R(A_1, ..., A_n)$ um esquema relacional. Denotamos por $Atr(R)$ o conjunto de atributos de $R = \{A_1, ..., A_n\}$. Para cada $A \in Atr(R)$, dom(A) é um conjunto finito de valores de A (o domínio de A). O conjunto $Tup(R) =$ dom$(A_1) \times$ dom$(A_2) \times ... \times$ dom(A_n) é o conjunto de todas as tuplas possíveis de R.

Uma ordem de preferência é definida da seguinte maneira:

Definição 3.3. (Ordem de Preferência) Uma *ordem de preferência* sobre $Tup(R)$ é uma relação binária $Pref$ sobre $Tup(R)$ (isto é, um subconjunto $Pref \subseteq Tup(R) \times Tup(R)$) satisfazendo as propriedades: (1) *irreflexiva* e (2) *transitiva*.

Logo, uma ordem de preferência é uma relação binária irreflexiva e transitiva sobre duas tuplas t_1 e t_2, que permite estabelecer se t_1 é preferida a t_2, se t_2 é preferida a t_1 ou se t_1 e t_2 são incomparáveis. No contexto da linguagem CPref-SQL, uma teoria-pc Γ sobre uma relação R induz uma *ordem de preferência* sobre as tuplas de R.

Neste trabalho, propomos duas maneiras distintas para uma consulta CPref-SQL obter uma ordem de preferência sobre um conjunto de tuplas: *ordem de preferência forte* e *ordem de preferência fraca*, ambas inspiradas no formalismo de [Wilson 2004], discutido no Capítulo 2.

A ordem forte é baseada na semântica *ceteris paribus*, mais conservativa, e, com isso, muitas tuplas tornam-se incomparáveis. Além disso, é a ordem mais intuitiva. Já a ordem fraca é mais relaxada e consegue obter muito mais pares de preferências; entretanto, é menos natural. O objetivo é poder oferecer alternativas para obtenção das tuplas preferidas, de forma que o usuário possa escolher a opção mais adequada às suas necessidades.

A seguir, definimos formalmente as propostas para cálculo da ordem de preferência. A cadeia de raciocínio que apresentaremos indica que a ordem fraca é um refinamento da ordem forte e que a proposta dessa ordem fraca surgiu a partir da necessidade de mostrar que a ordem forte (mais intuitiva) é, de fato, uma ordem de preferência.

Assim, temos: primeiramente, definimos a *ordem de preferência forte*. Entretanto, a ordem forte não é garantidamente uma ordem irreflexiva, logo, precisamos garantir que ela seja uma ordem de preferência. Para tanto, definimos uma *teoria-pc consistente* e baseamos na noção de que a ordem forte deve ser induzida a partir de uma teoria-pc consistente para ser irreflexiva. Na Subseção 3.1.3, apresentamos os fundamentos teóricos e a implementação do *teste de consistência* de uma teoria-pc. Inicialmente, introduzimos a *ordem de preferência fraca* sobre uma teoria-pc *localmente consistente* e com *grafo de dependência* acíclico e provamos que é uma ordem de preferência (irreflexiva e transitiva). Mostramos que a ordem forte está contida na ordem fraca e, portanto, é uma ordem de preferência. Obtemos, então, duas condições suficientes para que uma teoria-pc seja consistente, e sempre tenhamos ordens irreflexivas e transitivas: que a teoria-pc seja localmente consistente e seu grafo de dependência seja acíclico. Por fim, detalhamos a

implementação do teste de consistência baseado nessas duas condições obtidas.

Definição 3.4. (Ordem de Preferência Forte) Seja φ uma regra-pc sobre $Atr(R)$. Denotamos $>_\varphi$ o conjunto de pares de tuplas $(t_1, t_2) \in \text{Tup}(R)$ que satisfazem as seguintes condições:

- para cada atributo $A \in Atr(R)$ que aparece do lado esquerdo de φ, temos $t_1[A] = t_2[A]$ e $t_1[A] \models P_\varphi[A]$ (predicado P associado ao atributo A em φ);
- para cada atributo $B \in Atr(R)$ que não aparece em φ, temos $t_1[B] = t_2[B]$;
- para cada atributo X que aparece do lado direito de φ, temos $t_1[X] \models Q_1(X)$ e $t_2[X] \models Q_2(X)$.

Seja Γ uma teoria-pc sobre $Atr(R)$. Denotamos por $>_\Gamma$ o fecho transitivo de $\bigcup_{\varphi \in \Gamma} >_\varphi$. Γ induz uma ordem de preferência forte sobre $\text{Tup}(R)$. Logo, para todo par de tuplas $(t_1, t_2) \in >_\Gamma$, dizemos que t_1 é preferido a t_2 de acordo com uma ordem de preferência forte e denotamos, $t_1 >_\Gamma t_2$.

Exemplo 3.2. Considere o esquema relacional $R(A, B, C)$, para $dom(A) = \{a_1\}$, $dom(B) = \{b_1, b_2, b_3\}$ e $dom(C) = \{c_1, c_2\}$. Seja a teoria-pc $\Gamma = \{\varphi_1, \varphi_2\}$ sobre $Atr(R)$, onde:

$\varphi_1 : A = a_1 \rightarrow B = b_1 > B = b_2$;
$\varphi_2 : B = b_2 > B = b_3$ $[\{C\}]$.

De acordo com a ordem de preferência forte induzida por Γ, temos que:

- $>_{\varphi_1} = \{((a_1, b_1, c_1), (a_1, b_2, c_1)), ((a_1, b_1, c_2), (a_1, b_2, c_2))\}$,
- $>_{\varphi_2} = \{((a_1, b_2, c_1), (a_1, b_3, c_1)), ((a_1, b_2, c_1), (a_1, b_3, c_2)), ((a_1, b_2, c_2), (a_1, b_3, c_1)), ((a_1, b_2, c_2), (a_1, b_3, c_2))\}$ e
- $>_\Gamma = \{((a_1, b_1, c_1), (a_1, b_3, c_1)), ((a_1, b_1, c_1), (a_1, b_3, c_2)), ((a_1, b_1, c_2), (a_1, b_3, c_1)), ((a_1, b_1, c_2), (a_1, b_3, c_2))\} \cup >_{\varphi_1} \cup >_{\varphi_2}$.

Assim, dizemos, por exemplo, que a tupla $t_1 = (a_1, b_1, c_1)$ é preferida a $t_2 = (a_1, b_3, c_1)$, ou seja: $t_1 >_\Gamma t_2$.

Note, pela Definição 3.4, que para cada regra-pc φ, a relação $>_\varphi$ mantém uma semântica *ceteris paribus* (todo o restante igual) em relação aos atributos que não pertencem à regra φ (vide Subseção 2.1.1). Ou seja, tuplas com valores *diferentes* para cada atributo $B \in Atr(R)$, que *não* está em φ, são incomparáveis de acordo com $>_\varphi$.

Retomando ao contexto do Exemplo 3.1, de acordo com a semântica da ordem forte de preferência, as regras da teoria-pc Γ expressam os seguintes desejos do usuário:

φ_1**:** Em geral, entre viagens de mesma categoria, prefiro aquelas com preço até R$3000;

φ_2: Para viagens de mesmo preço, prefiro roteiros urbanos a cruzeiros;

φ_3: Entre viagens de roteiro urbano de mesmo preço, prefiro aquelas nacionais.

Logo, o significado da regra-pc φ_1 do Exemplo 3.1 é o seguinte: dadas duas tuplas t_1 e t_2, se elas tiverem a mesma categoria e o preço de t_1 for < 3000 e o preço de t_2 for ≥ 3000, então t_1 é preferida a t_2, ou seja, $t_1 >_{\varphi_1} t_2$. Veja que os atributos entre colchetes representam aqueles atributos cujos valores não interferem na escolha das preferidas. Como o atributo `Categoria` não é antecedente, nem consequente e nem está entre colchetes, para que duas tuplas possam ser comparadas através dessa regra φ_1, elas têm que ser da mesma categoria (*ceteris paribus*).

Ouro exemplo: em φ_2, dadas duas tuplas t_1 e t_2, onde t_1 é um roteiro urbano e t_2 é uma opção de cruzeiro, se ambas tiverem exatamente o mesmo preço, então $t_1 >_{\varphi_2} t_2$.

Finalmente, de acordo com φ_3, se duas tuplas t_1 e t_2 tiverem o roteiro urbano, independente dos valores de `Destino` e `Duração`, e desde que sejam do mesmo preço (*ceteris paribus*), então aquela de categoria nacional é preferida à internacional.

A ordem forte induzida sobre as tuplas de `Viagens` pela teoria-pc Γ do Exemplo 3.1 está ilustrada pelo grafo *better-than* da Figura 3.2. Na figura, uma seta de t para t' significa que $t >_\Gamma t'$. Setas obtidas pelo fecho transitivo não estão representadas na figura. Note, por exemplo, que $t_2 > t_4$ pela regra φ_1, $t_4 > t_5$ pela regra φ_3 e, por transitividade, $t_2 > t_5$.

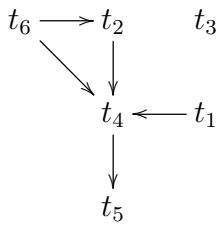


Figura 3.2: Grafo *better-than* da relação `Viagens` sob a ordem forte de preferência.

A partir da Definição 3.4 podemos inferir dois fatos importantes:

(1) cada regra φ individualmente induz uma ordem sobre as tuplas; nessa ordem, as *preferências locais* sobre os valores dos atributos das tuplas não são *absolutas*, ou seja, só é possível inferir que uma tupla é preferida a outra em relação ao atributo consequente de φ, dependendo de como os outros atributos estão instanciados;

(2) para testar se duas tuplas $t = (a_1, a_2, ..., a_n)$ e $t' = (b_1, b_2, ..., b_n)$ são comparáveis de acordo com a teoria-pc $\Gamma = \{\varphi_1, ..., \varphi_k\}$, ou seja, $t >_\Gamma t'$, devemos verificar se existe um subconjunto de $\Gamma' = \{\psi_1, ..., \psi_m\}$ para $\Gamma' \subseteq \Gamma$, e tuplas $s_1, ..., s_{m-1}$ tal que $t >_{\psi_1} s_1, s_1 >_{\psi_2} s_2, \ ... \ , s_{m-1} >_{\psi_m} t'$.

Em síntese, a ordem forte de preferência do modelo da CPref-SQL não induz uma preferência local absoluta sobre os atributos (daí preferências condicionais) e é inferida a

partir do fecho transitivo da união das ordens de preferência de cada regra-pc. Isso implica que a ordem de preferência induzida por uma teoria-pc é transitiva por definição, mas não é irreflexiva, ou seja, ela não é necessariamente uma *ordem parcial estrita*. Durante uma consulta com preferência condicional, pode ser inferido que "prefiro X a X!" De fato, $>_\Gamma$ pode conter um par (t, t), o que significa que $>_\Gamma$ não necessariamente satisfaz a propriedade de ser irreflexiva. Logo, a ordem forte não é necessariamente uma ordem de preferência.

Como reflexibilidade numa ordem de preferência não é desejável no contexto de banco de dados, questões de *consistência* de uma teoria-pc devem ser tratadas cuidadosamente para evitar que situações em que uma tupla é preferida a ela mesma sejam inferidas durante uma consulta-pc.

Fazendo um estudo comparativo com a ordem de *preferência Pareto* $\otimes$, que é um outro tipo de composição de preferências, utilizada nas consultas *Skyline* [Börzsönyi et al. 2001] e nas consultas Pareto de [Kießling and Köstler 2002], conforme apresentamos no Capítulo 2, nessa abordagem uma situação reflexiva nunca pode ser inferida.

Uma ordem de preferência Pareto é definida da seguinte maneira: $t \otimes t'$ se existe $i \in \{1, ..., n\}$ tal que $a_i > b_i$ e para todo $j \neq i$, tem-se $a_j \not> b_j$. Note que nesse caso, as preferências locais sobre os valores dos atributos das tuplas são absolutas: comparamos valores do i-ésimo atributo de t com os valores do i-ésimo atributo de t'. Ou seja, as consultas *Skyline* e Pareto não precisam levar em conta questões de inconsistência de preferências, pois o resultado da acumulação Pareto de ordens fracas sobre os domínios dos atributos é uma ordem parcial estrita sobre o conjunto de tuplas (para prova, ver [Chomicki 2003]).

No contexto da ordem forte da linguagem CPref-SQL, buscamos, então, por teorias-pc consistentes para garantir que tal ordem seja uma ordem de preferência, conforme formalizamos na definição a seguir.

Definição 3.5. (Teoria-pc consistente) Dizemos que uma teoria-pc Γ é consistente se e somente se a ordem induzida $>_\Gamma$ é irreflexiva.

3.1.3 Teste de Consistência

Na busca por condições suficientes para que uma teoria-pc seja consistente, inicialmente introduzimos a noção de ordem de preferência fraca.

A motivação para inserirmos a ordem fraca no modelo de preferência da CPref-SQL é que, nessa abordagem, as preferências induzem uma ordem menos conservativa, mais relaxada. Aqui, a semântica *ceteris paribus* não é levada em conta. Denominada também como *ordem de preferência lexicográfica condicional parcial* em [Wilson 2004], ela é baseada na relação de importância entre atributos.

Basicamente, para comparar duas tuplas que possuem valores diferentes em vários atributos, leva-se em conta apenas os valores dos atributos considerados mais importantes.

A definição formal da ordem fraca na linguagem CPref-SQL baseia-se na noção de *grafo de dependência preferencial* entre atributos e *consistência local* de uma teoria-pc, cujas definições são análogas às Definições 2.3 e 2.4 de [Wilson 2004], conforme mostramos a seguir.

Definição 3.6. (Grafo de Dependência Preferencial) Dado um conjunto de atributos V e uma teoria-pc Γ sobre V, $G(\Gamma)$ denota o grafo de dependência preferencial de Γ. $G(\Gamma)$ é um grafo dirigido definido como $G(\Gamma) = (V, E)$, onde V é o conjunto de atributos em Γ e $E = \bigcup_{\varphi_i \in \Gamma} E(\varphi_i)$, sendo que $E(\varphi_i) = \{(Y, X_{\varphi_i}) \mid Y \in Atr_{\varphi_i}(u)\} \cup \{(X_{\varphi_i}, Z) \mid Z \in W_{\varphi_i}\}$.

Exemplo 3.3. Como exemplo, considere a seguinte teoria-pc Γ sobre os atributos da relação `Viagens(D,R,P,Du,C)`:

$$\Gamma = \left\{ \begin{array}{l} \varphi_1 : R = urb > R = cruz\ [\{D, Du, C\}], \\ \varphi_2 : C = nac \rightarrow R = eco > R = praia\ [\{Du\}]. \end{array} \right\}$$

A Figura 3.3 ilustra G(Γ) de acordo com a Definição 3.6. Neste exemplo, G(Γ) é cíclico, indicando que o atributo C depende do atributo R e vice-versa.

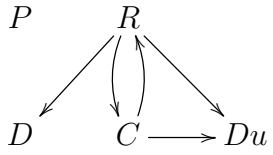


Figura 3.3: Grafo de dependência preferencial da teoria-pc Γ.

Definição 3.7. (Consistência Local) Seja Γ uma teoria-pc sobre um conjunto de atributos V e $X \in V$. Denotamos Γ_X o conjunto de regras-pc $\varphi \in \Gamma$ onde X aparece do lado direito da regra. Seja $V' \subseteq V$ o conjunto de atributos que aparecem do lado esquerdo das regras-pc de Γ_X e $u \in Tup(V')$. Vamos considerar o conjunto de pares $\mathcal{R}_u^X = \{(x_1, x_2) \in \mathbf{dom}(X) \times \mathbf{dom}(X) \mid$ existe $\varphi \in \Gamma_X$ tal que u é compatível com φ e $x_i \models Q_i(X)$ do lado direito de φ, para $i = 1, 2\}$. Definimos a relação binária $\succ_u^X$ como o fecho transitivo de $\mathcal{R}_u^X$.

Dizemos que Γ é localmente consistente no atributo X se $\succ_u^X$ é irreflexiva para todo $u \in Tup(V')$. Dizemos que Γ é *localmente consistente* se ela é localmente consistente em cada atributo X que aparece do lado direito de suas regras-pc.

Exemplo 3.4. Para exemplificar, considere a seguinte teoria-pc Γ:

$$\Gamma = \left\{ \begin{array}{l} \varphi_1 : R = urb > R = cruz\ [\{D, Du\}], \\ \varphi_2 : C = nac \rightarrow R = cruz > R = praia\ [\{D\}], \\ \varphi_3 : P < 4000 \rightarrow R = praia > R = urb\ [\{D, Du\}]. \end{array} \right\}$$

Constatamos que Γ não é localmente consistente, pois não há consistência local no atributo R. A relação $\succ_u^R$, quando $u = (\text{C=nac}) \wedge (\text{P} < 4000)$, produz os seguintes pares: {(urb,cruz), (cruz,praia), (praia,urb), (cruz,urb), (urb,praia), **(urb,urb)**, **(praia,praia)**, **(cruz,cruz)**}, sendo os três últimos reflexivos.

Basicamente, então, G(Γ) é o grafo que relaciona a dependência entre os atributos da teoria-pc Γ. G(Γ) ser acíclico significa que não existe nenhuma situação em que o valor de um atributo A depende do valor do atributo B e, ao mesmo tempo, o valor de B depende de A. Já a consistência local está relacionada com a ordem local induzida no valor de um atributo. Ser localmente consistente significa que não existe nenhuma possibilidade da ordem sobre o valor a_1 de um atributo A ser do tipo: a_1 é preferida a a_1.

Definição 3.8. (Ordem de Preferência Fraca) Seja Γ uma teoria-pc *localmente consistente* sobre o conjunto de atributos V, tal que $G(\Gamma)$ é *acíclico*.

Seja α, $\beta \in \text{Tup}(V)$. Definimos $\Delta(\alpha, \beta)$ o conjunto de atributos em V onde α e β se diferem, ou seja, $\{X \in V \mid \alpha[X] \neq \beta[X]\}$.

Para um subconjunto W de nós de G, denotamos inf(W) o conjunto $\{X \in W |$ não existem ancestrais de X (em relação a G) em $W\}$.

Uma tupla α é dita *preferida* a β de acordo com Γ (denotado por $\alpha \succ_\Gamma \beta$) se para todo $X \in \text{inf}(\Delta(\alpha, \beta))$ tem-se $(\alpha[X], \beta[X]) \in \succ_\alpha^X \cap \succ_\beta^X$.

Retomando o Exemplo 3.1, agora a semântica das regras da teoria Γ é diferente, um pouco mais relaxada. É como se o usuário expressasse seus desejos da seguinte maneira:

φ_1**:** Em geral, prefiro viagens com o preço menor que R$3000, sendo o preço, mais importante que o destino, roteiro e duração;

φ_2**:** Em relação ao roteiro, prefiro viagens urbanas a cruzeiros, independente do destino, duração e categoria;

φ_3**:** Prefiro viagens urbanas nacionais do que roteiros urbanos internacionais, independente do destino e duração.

Assim, a regra φ_1, por exemplo, significa que dadas duas tuplas t_1 e t_2, se t_1 tiver um preço < 3000 e t_2 custar ≥ 3000, independente dos valores de destino, duração e roteiro, então $t_1 \succ_{\varphi_1} t_2$.

Já a regra φ_2 indica que o usuário prefere a opção de um roteiro urbano do que um cruzeiro, e que o roteiro é mais relevante do que o destino, a duração e a categoria da viagem.

Por fim, de acordo com φ_3, entre duas tuplas t_1 e t_2 de roteiro urbano, se t_1 for nacional e t_2 internacional, então t_1 é preferida a t_2, para quaisquer destino e duração.

A Figura 3.4 ilustra o grafo da relação de dependência entre os atributos da teoria-pc $\Gamma = \{\varphi_1, \varphi_2, \varphi_3\}$ do Exemplo 3.1, de acordo com a definição.

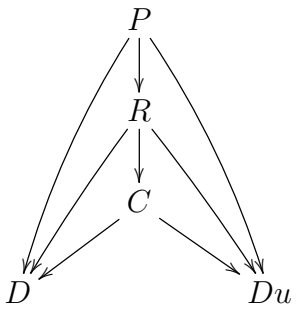


Figura 3.4: Grafo de dependência preferencial da teoria-pc Γ.

A comparação para obter uma ordem fraca entre duas tuplas α e β é feita com o cálculo dos conjuntos Δ e inf(Δ), conforme formalizado na Definição 3.8. O objetivo é comparar α e β apenas naqueles atributos cujos valores se diferem e que são considerados mais importantes de acordo com a teoria-pc.

Exemplo 3.5. Como exemplo, suponha que queiramos comparar as tuplas t_1 e t_4 da instância de `Viagens`, ilustrada na Figura 3.1. Então, temos:

- $\Delta(t_1, t_4) = \{\text{D, R, P, Du}\}$, representando o conjunto de atributos cujos valores se diferem em t_1 e t_4;
- $\text{inf}(\Delta(t_1, t_4)) = \{\text{P}\}$, que corresponde ao conjunto de atributos de Δ que não têm ancestrais em Δ de acordo com o grafo de dependência $G(\Gamma)$;
- $\mathcal{R}^P_{t_1} = \mathcal{R}^P_{t_4} = \{(x_1, x_2) \mid x_1 < 3000 \text{ e } x_2 \geq 3000\}$;
- Como o par $(t_1[P], t_4[P]) = (2000, 4000) \in \ \succ^P_{t_1} \cap \succ^P_{t_4}$, então $t_1 \succ_\Gamma t_4$.

A Tabela 3.2 ilustra os conjuntos Δ e inf(Δ) e o resultado das comparações entre as tuplas da relação `Viagens`, de acordo com a teoria-pc Γ. Omitimos o símbolo Γ da notação de ordem de preferência $\succ_\Gamma$, considerando que todas as comparações são referentes à teoria-pc Γ e denotamos $t \sim t'$ para indicar que as tuplas t e t' são incomparáveis em Γ.

Finalizando, a Figura 3.5 é o grafo *better-than* que sintetiza a ordem fraca induzida entre as tuplas de `Viagens`. Na figura, uma seta de t para t' significa que $t \succ_\Gamma t'$.

De acordo com a Definição 3.3, uma relação de ordem de preferência deve ser uma relação de ordem parcial estrita, ou seja, irreflexiva e transitiva. Através do Teorema 3.1 a seguir, é possível provar que a ordem de preferência fraca é uma relação de ordem de preferência.

t e t'	$\Delta(t,t')$	inf($\Delta(t,t')$)	Ordem Fraca
t_1 e t_2	{D,R,P,Du}	{P}	$t_1 \sim t_2$
t_1 e t_3	{D,R,P,C}	{P}	$t_1 \succ t_3$
t_1 e t_4	{D,R,P,Du}	{P}	$t_1 \succ t_4$
t_1 e t_5	{D,R,P,Du,C}	{P}	$t_1 \succ t_5$
t_1 e t_6	{D,R,P,Du}	{P}	$t_1 \sim t_6$
t_2 e t_3	{D,R,P,Du,C}	{P}	$t_2 \succ t_3$
t_2 e t_4	{D,R,P,Du}	{P}	$t_2 \succ t_4$
t_2 e t_5	{D,R,P,Du,C}	{P}	$t_2 \succ t_5$
t_2 e t_6	{D,R,Du}	{R}	$t_6 \succ t_2$
t_3 e t_4	{D,R,P,Du,C}	{P}	$t_3 \sim t_4$
t_3 e t_5	{D,R,P,Du}	{P}	$t_3 \sim t_5$
t_3 e t_6	{D,R,P,Du,C}	{P}	$t_6 \succ t_3$
t_4 e t_5	{D,Du,C}	{C}	$t_4 \succ t_5$
t_4 e t_6	{D,P,Du}	{P}	$t_6 \succ t_4$
t_5 e t_6	{D,P,Du,C}	{P}	$t_6 \succ t_5$

Tabela 3.2: Comparação entre as tuplas da relação `Viagens`

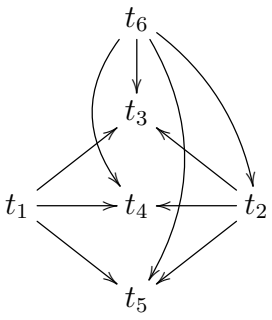


Figura 3.5: Grafo *better-than* da relação `Viagens` sob a ordem fraca de preferência.

Teorema 3.1. *A relação binária $\succ_\Gamma$ é uma ordem parcial estrita sobre Tup(V).*

Prova. Devemos mostrar que a relação de ordem fraca $\succ_\Gamma$ é irreflexiva e transitiva.

Irreflexiva. Ser irreflexiva significa que não é possível que, dada uma tupla α e uma teoria-pc Γ, $\alpha \succ_\Gamma \alpha$. Pela definição de ordem fraca, temos que:

- Dadas duas tuplas α e β, $\alpha \succ_\Gamma \beta$ se para todo $A \in inf(\Delta(\alpha,\beta))$ temos o par ($\alpha[A]$, $\beta[A]$) $\in \succ_\alpha^A \cap \succ_\beta^A$, ou seja, se para todo atributo $A \in inf$, $\alpha[A] \succ \beta[A]$ de acordo com a ordem local sobre o atributo A.
- A ordem fraca é definida sob teorias-pc localmente consistentes, ou seja: para todo atributo A, $\succ_\alpha^A$ é irreflexiva.
- Se $\succ_\alpha^A$ é irreflexiva, então um par ($\alpha[A]$, $\alpha[A]$) nunca fará parte da relação $\succ_\alpha^A \cap \succ_\alpha^A$.
- Logo, α não é preferida a α e a ordem fraca é irreflexiva.

Transitiva. Sejam as tuplas α, β e γ e uma teoria-pc Γ. Queremos mostrar que se $\alpha \succ_\Gamma \beta$ e $\beta \succ_\Gamma \gamma$, então $\alpha \succ_\Gamma \gamma$.

- Devemos mostrar que $\forall A \in inf(\Delta(\alpha,\gamma))$, $\alpha \succ_t^A \gamma$, para todo $t \in Tup(Pa_H(A))$. $Pa_H(A)$ são os pais do atributo A em relação ao grafo H (o grafo de dependência preferencial entre os atributos que aparecem em Γ).
- Como $A \in inf(\Delta(\alpha,\gamma))$ temos que: para todo ancestral C de A, $\alpha[C] = \gamma[C]$ e $\alpha[A] \neq \gamma[A]$.
- Logo, duas situações podem ocorrer: **Caso 1:** $\alpha[A] \neq \beta[A]$ ou **Caso 2:** $\beta[A] \neq \gamma[A]$.

  Caso 1: $\alpha[A] \neq \beta[A]$. Vamos mostrar que para todo ancestral C de A, $\alpha[C] = \beta[C]$.

  - Por contradição, suponha que $\alpha[C] \neq \beta[C]$.
  - Sabemos que $\alpha[C] = \gamma[C]$.
  - Logo, $\beta[C] \neq \gamma[C]$.
  - Sem perda de generalidade, suponha que $\forall D$ ancestral de C, $\alpha[D] = \beta[D]$.
  - Como, por hipótese, $\alpha \succ_\Gamma \beta$, então $C \in inf(\Delta(\alpha,\beta))$, ou seja, $\alpha[C] \succ \beta[C]$.
  - Assim, concluímos que $\gamma[C] \succ \beta[C]$, o que é contradição, pois, por hipótese, $\beta \succ_\Gamma \gamma$ e, como, $\alpha[D] = \gamma[D] = \beta[D]$, $C \in inf(\Delta(\beta,\gamma))$ e $\beta[C] \succ \gamma[C]$.

  Mostramos que $\alpha[A] \neq \beta[A]$ e que para todo ancestral C de A, $\alpha[C] = \beta[C]$. Logo, $A \in inf(\Delta(\alpha,\beta))$. Duas situações podem ocorrer: **Caso 1.1:** $\beta[A] = \gamma[A]$ ou **Caso 1.2:** $\beta[A] \neq \gamma[A]$.

  - **Caso 1.1:** $\beta[A] = \gamma[A]$. Como, por hipótese, $\alpha \succ_t^A \beta$. Então, $\alpha \succ_t^A \gamma$, c.q.d.
  - **Caso 1.2:** $\beta[A] \neq \gamma[A]$. Utilizando o mesmo raciocínio para prova do Caso 2, podemos concluir que $A \in min(\Delta(\beta,\gamma))$. Como, por hipótese, $\beta \succ_\Gamma \gamma$, então $\beta \succ_t^A \gamma$. Temos também que $\alpha \succ_t^A \beta$. Como, por definição, a relação $\succ_t^A$ é o fecho transitivo, então, $\alpha \succ_t^A \gamma$, c.q.d.

  Caso 2: $\beta[A] \neq \gamma[A]$. Vamos mostrar que para todo ancestral C de A, $\beta[C] = \gamma[C]$.

  - Por contradição, suponha que $\beta[C] \neq \gamma[C]$.
  - Sabemos que $\alpha[C] = \gamma[C]$.
  - Logo, $\beta[C] \neq \alpha[C]$.
  - Sem perda de generalidade, suponha que $\forall D$ ancestral de C, $\beta[D] = \gamma[D]$.
  - Como, por hipótese, $\beta \succ_\Gamma \gamma$, então $C \in inf(\Delta(\beta,\gamma))$, ou seja, $\beta[C] \succ \gamma[C]$.
  - Assim, concluímos que $\beta[C] \succ \alpha[C]$, o que é contradição, pois, por hipótese, $\alpha \succ_\Gamma \beta$ e, como, $\beta[D] = \alpha[D] = \gamma[D]$, $C \in inf(\Delta(\alpha,\beta))$ e $\alpha[C] \succ \beta[C]$.

  Mostramos que $\beta[A] \neq \gamma[A]$ e que para todo ancestral C de A, $\beta[C] = \gamma[C]$. Logo, $A \in inf(\Delta(\beta,\gamma))$. Duas situações podem ocorrer: **Caso 2.1:** $\alpha[A] = \beta[A]$ ou **Caso 2.2:** $\alpha[A] \neq \beta[A]$.

- **Caso 2.1:** $\alpha[A] = \beta[A]$. Como, por hipótese, $\beta \succ_t^A \gamma$. Então, $\alpha \succ_t^A \gamma$, c.q.d.
- **Caso 2.2:** $\alpha[A] \neq \beta[A]$. Utilizando o mesmo raciocínio para prova do Caso 1, podemos concluir que $A \in min(\Delta(\alpha, \beta))$. Como, por hipótese, $\alpha \succ_\Gamma \beta$, então $\alpha \succ_t^A \beta$. Temos também que $\beta \succ_t^A \gamma$. Como, por definição, a relação $\succ_t^A$ é o fecho transitivo, então, $\alpha \succ_t^A \gamma$, c.q.d.

Conforme discutimos anteriormente, a relação de ordem fraca é menos conservativa do que a ordem de preferência forte. Comparando as Figuras 3.2 e 3.5, percebemos que na abordagem da ordem fraca há uma quantidade muito maior de relações estabelecidas entre as tuplas.

É possível mostrar que a ordem forte está contida na ordem fraca, ou seja, a ordem fraca é um refinamento da ordem forte. O Teorema 3.2 formaliza essa relação entre as duas ordens no contexto do modelo de preferências generalizado da CPref-SQL.

Teorema 3.2. *Seja* Γ *uma teoria-pc sobre o conjunto de atributos* V*. Então, a ordem forte está contida na ordem fraca (ordem lexicográfica parcial), ou seja:* $>_\Gamma \subseteq \succ_\Gamma$.

Prova. A prova deste teorema é consequência imediata do Teorema 3.1. Se a ordem fraca é uma relação (binária) irreflexiva e transitiva, sobre uma teoria-pc localmente consistente e com grafo de dependência acíclico, então é possível transformar uma relação de ordem forte em ordem fraca, apenas inserindo no conjunto de atributos entre colchetes de cada regra-pc φ de uma teoria-pc Γ, os atributos descendentes daqueles que aparecem no antecedente e consequente de φ. Para detalhes, ver [Wilson 2004].

A partir do Teorema 3.2, podemos concluir que todo par de tupla que pode ser comparado usando a ordem forte também pode ser comparado da mesma maneira usando a ordem fraca. Entretanto, a recíproca não é verdadeira. Daí a motivação para a proposta de dois tipos de ordem de preferência: (1) a ordem forte: mais natural, porém mais restritiva e (2) a ordem fraca: menos restritiva, porém menos natural.

Diante da definição do modelo de preferência da CPref-SQL, vale destacar que as ordens de preferência induzidas sobre as tuplas do banco de dados dependem apenas da teoria-pc em questão. Dizer se uma tupla é preferida a outra não depende, portanto, da instância do banco de dados.

Por fim, o Teorema a seguir foi proposto em [Wilson 2004], para o modelo de preferência sem extensões de predicados (Teorema 2.1). Entretanto, podemos utilizá-lo também para o contexto da teoria-pc estendida da CPref-SQL, pois ele é consequência imediata do Teorema 3.2, no qual mostramos que $>_\Gamma \subseteq \succ_\Gamma$, onde Γ é uma teoria-pc estendida da CPref-SQL.

Teorema 3.3. ([Wilson 2004]) *Seja* Γ *uma teoria-pc localmente consistente sobre um conjunto de atributos* V *tal que* $G(\Gamma)$ *é acíclico. Então* Γ *é consistente.*

Implementação do Teste de Consistência

Para garantir, então, que apenas teorias-pc consistentes sejam submetidas ao mecanismo da CPref-SQL, implementamos um teste de consistência. A Figura 3.6 descreve o algoritmo **ConsistencyTest** que é baseado nas conclusões do Teorema 3.3.

ConsistencyTest(Γ) :
Input: uma teoria-pc Γ
Output: **true**: se Γ é consistente
false: se Γ não é consistente

```
1: if CheckDependencyGraph(Γ) and CheckLocalConsistency(Γ) then
2:     return true
3: return false
```

Figura 3.6: Algoritmo **ConsistencyTest**

A rotina *CheckDependencyGraph* tem como entrada uma teoria-pc Γ e retorna *true* se G(Γ) é acíclico. Caso contrário, retorna *false*, conforme detalhado na Figura 3.7.

CheckDependencyGraph(Γ) :
Input: uma teoria-pc Γ
Output: **true**: se G(Γ) é acíclico
false: se G(Γ) é cíclico

```
1: V = ∅
2: E = ∅
3: for all φ_i ∈ Γ do
4:       // Atr(u)_φi = conjunto dos atributos antecedentes de φ_i
5:       // X_φi = atributo consequente de φ_i
6:       // W_φi = conjunto dos atributos entre colchetes de φ_i
7:     V = V ∪ {x | x ∈ Atr(u)_φi ∪ {X_φi} ∪ W_φi}
8:     E = E ∪ {(A, X) | A ∈ Atr(u)_φi}
9:     E = E ∪ {(A, C) | A ∈ Atr(u)_φi, C ∈ W_φi}
10:    E = E ∪ {(X, C) | C ∈ W_φi}
11: G_Γ = (V, E)                          // grafo de dependência de Γ
12: if cyclic (G_Γ) then
13:     return false
14: return true
```

Figura 3.7: Algoritmo **CheckDependencyGraph**

A rotina *CheckLocalConsistency* verifica a consistência local da teoria-pc Γ e seu algoritmo é apresentado na Figura 3.8. Basicamente, ele seleciona o conjunto de regras-pc que têm o mesmo atributo consequente e a subrotina *datalogP* calcula a consistência local através de um programa Datalog P.

O objetivo da função *datalogP* é construir P e submeter o *goal* para uma *engine* Datalog. Se o *goal* for satisfeito, significa que existe uma inconsistência: *datalogP* retorna *true* indicando que o *goal* foi satisfeito e *CheckLocalConsistency* retorna *false*.

CheckLocalConsistency(Γ) :

Input: uma teoria Γ
Output: **true**: se Γ é localmente consistente
false: se Γ não é localmente consistente

```
1: for all rule φ_i ∈ Γ do
2:     if φ_i não foi analisado then
3:         Θ_X ← ∅
4:         for all rule φ_j ∈ Γ do
5:             if X_φj = X_φi then          // X_φ é o atributo consequente de φ
6:                 Θ_X = Θ_X ∪ {φ_j}        // analisa φ_j
7:         if datalogP(Θ_X) = true then
8:             return false
9: return true
```

Figura 3.8: Algoritmo **CheckLocalConsistency**

Considerando que a implementação da CPref-SQL neste trabalho suporta os predicados $>, <, =, \neq, \geq$ e $\leq$, para efeitos de codificação em Datalog, os predicados das regras-pc são tratados como intervalos. O mapeamento é feito da seguinte maneira: seja $P(A)$ um predicado unário sobre o atributo A da forma: $A\theta a$, onde $a \in \mathbf{dom}(A)$, $\theta \in \{<, >, =, \neq, \geq, \leq\}$. Então, temos:

- $A = a \equiv [a, a]$
- $A > a \equiv (a, \infty)$
- $A < a \equiv (-\infty, a)$
- $A \neq a \equiv (-\infty, a) \cup (a, \infty)$
- $A \geq a \equiv [a, \infty)$
- $A \leq a \equiv (-\infty, a]$

Assim, P é construído e executado pela função *datalogP*(Θ_X) como se segue:

(1) para cada regra-pc $r \in \Theta_X$ da forma: $u \rightarrow Q_1(X) > Q_2(X)$, para $u = P_1(A_1) \wedge ... \wedge P_k(A_k)$, é associada uma cláusula de Horn p_r:

$$pref(Y_1, ..., Y_n, X_1, X_2) \leftarrow Y_1 \equiv P_1(A_1), ..., Y_k \equiv P_k(A_k),$$
$$X_1 \equiv Q_1(X),\ X_2 \equiv Q_2(X).$$

onde $Y_1, ..., Y_n, X_1, X_2$ são intervalos, $\{Y_1, ..., Y_k\} \subseteq \{Y_1, ..., Y_n\}$ e $\{A_1, ..., A_n, X\}$ é o conjunto de todos os atributos pertencentes às regras de Θ_X.

(2) são consideradas mais duas cláusulas, responsáveis pelo fecho transitivo:

$$\begin{aligned} dom(Y_1,...,Y_n,X_1,X_2) \leftarrow\ & pref(AY_1,...,AY_n,X_1,Z),\\ & inter(Y_1,AY_1,-),\\ & ...,\\ & inter(Y_n,AY_n,-),\\ & inter(X_2,Z,-). \end{aligned}$$

$$\begin{aligned} dom(Y_1,...,Y_n,X_1,X_2) \leftarrow\ & pref(AY_1,...,AY_n,X_1,Z_1),\\ & inter(Y_1,AY_1,RY_1),\\ & ...,\\ & inter(Y_n,AY_n,RY_n),\\ & pref(-,...,-,Z,-),\\ & inter(Z_1,Z,-),\\ & dom(RY_1,...,RY_n,Z,X_2). \end{aligned}$$

onde: $Y_1,...,Y_n,X_1$ e X_2 são intervalos e o predicado *inter* é definido da seguinte maneira: sejam A, B e C intervalos, então *inter(A, B, C)* é satisfeito se: $A \cap B = C$ e $C \neq \emptyset$.

(3) uma última cláusula que servirá como *goal* é criada:

$$inconsistent(A) \leftarrow dom(-,...,-,A,A).$$

O teste é obtido pela execução do $goal \leftarrow inconsistent(A)$, onde A é um intervalo. Se A unificar com um intervalo, o *goal* é **true** e significa que: $\exists\varphi \in \Theta_X \mid Q_{1_\varphi}(X) \equiv A$ e $\exists a \in A \mid a >_u^X a$. Caso contrário o *goal* é **false**.

Logo, o objetivo de P é encontrar algum elemento pertencente aos intervalos consequentes das regras de Θ_X que é preferido a ele mesmo. Ou seja, é possível que exista uma tupla t compatível com todas as regras-pc pertencentes a Θ_X, tal que $t \succ_\Gamma t$?

Exemplo 3.6. Como exemplo, considere a teoria-pc Γ do Exemplo 3.4, sobre a relação `Viagens(D,R,P,Du,C)`:

$$\Gamma = \left\{ \begin{array}{l} \varphi_1 : R = urb > R = cruz\ [\{D, Du\}],\\ \varphi_2 : C = nac \rightarrow R = cruz > R = praia\ [\{D\}],\\ \varphi_3 : P < 4000 \rightarrow R = praia > R = urb\ [\{D, Du\}]. \end{array} \right\}$$

A rotina *CheckDependencyGraph* construirá o grafo de dependência de Γ, ilustrado pela Figura 3.9 e verificará que ele é acíclico, retornando **true**.

Já para calcular a consistência local de Γ, *CheckLocalConsistency* construirá o conjunto Θ_R e *datalogP(Θ_R)* gerará o seguinte programa P:

$$pref(D,P,Du,C,R1,R2) \leftarrow\ R_1 \equiv [urb,urb],\ R_2 \equiv [cruz,cruz].$$
$$pref(D,P,Du,C,R1,R2) \leftarrow\ C \equiv [nac,nac],\ R_1 \equiv [cruz,cruz],\ R_2 \equiv [praia,praia].$$

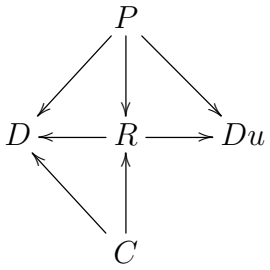


Figura 3.9: Grafo de dependência preferencial da teoria-pc Γ.

$pref(D, P, Du, C, R1, R2) \leftarrow\ P \equiv (-\infty, 4000),\ R_1 \equiv [praia, praia],\ R_2 \equiv [urb, urb].$

$$
\begin{aligned}
dom(D, P, Du, C, R1, R2) \leftarrow\ & pref(D1, P1, Du1, C1, R1, Z), \\
& inter(D, D1, -), \\
& inter(P, P1, -), \\
& inter(Du, Du1, -), \\
& inter(C, C1, -), \\
& inter(R2, Z, -). \\
dom(D, P, Du, C, R1, R2) \leftarrow\ & pref(D1, P1, Du1, C1, R1, Z1), \\
& inter(D, D1, RD1), \\
& inter(P, P1, RP1), \\
& inter(Du, Du1, RDu1), \\
& inter(C, C1, RC1), \\
& pref(-, ..., -, Z, -), \\
& inter(Z1, Z, -), \\
& dom(RD1, RP1, RDu1, RC1, Z, R2).
\end{aligned}
$$

$inconsistent(A) \leftarrow dom(-, ..., -, A, A).$

Ao executar o *goal* ← *inconsistent(A)*, A unificará com o intervalo $[urb, urb]$, indicando que o roteiro *urb* é preferido a ele mesmo. Logo, temos uma inconsistência, e *CheckLocalConsistency* retorna **false**. Ou seja, Γ não é consistente.

Detalhes de Implementação

Nosso objetivo com a implementação do teste de consistência é fornecer uma ferramenta para cálculo da consistência de uma teoria-pc sob o modelo da CPref-SQL. A ferramenta está integralmente implementada em código C. Em especial, desenvolvemos uma *engine* que implementa o método da resolução SLD [Lloyd 1984], responsável pelos cálculos de recursão, *backtracking* e unificação.

Atualmente o teste não está integrado diretamente ao sistema de consultas. Ao criar um conjunto de preferências no banco de dados, através da CPref-SQL, presume-se que elas já foram testadas e são consistentes. Entretanto, já construímos o teste de consistência em linguagem totalmente compatível com a implementação da CPref-SQL, de forma que

ele possa ser integrado posteriormente. Tal integração é, portanto, um potencial trabalho futuro.

3.2 CPref-SQL: uma extensão SQL

Nesta seção apresentamos os operadores da álgebra CPref-SQL, em especial o operador ***SelectK-Best*** que é uma contribuição do nosso trabalho. Uma vez definidos os operadores que estendem a álgebra SQL, detalhamos a sintaxe da linguagem CPref-SQL com o novo operador, bem como o plano canônico de execução de uma consulta com preferências condicionais (consulta-pc).

3.2.1 Operadores da Álgebra CPref-SQL

A álgebra CPref-SQL é formada pelos operadores ***Select-Best*** e ***SelectK-Best***.

SelectK-Best$_\Gamma(K,R)$ ou simplesmente ***SelectK-Best***(K,R) onde Γ é conhecido no contexto, é o operador que seleciona o conjunto das top-K tuplas de uma dada relação R de acordo com uma teoria-pc Γ. Para definirmos formalmente a semântica do ***SelectK-Best***, é necessário introduzirmos a noção de *nível* de uma tupla.

Definição 3.9. (Nível) Seja Γ uma teoria-pc sobre um esquema relacional $R(A_1, ..., A_n)$, r uma instância de banco de dados sobre R e t uma tupla $\in r$. O *nivel* de t, denotado $n(t)$, de acordo com Γ é definido por indução da seguinte maneira:

se $\nexists\ t' \in r$ tal que $t'\ \rho\ t$, para $\rho \in \{\succ_\Gamma, >_\Gamma\}$, definimos $n(t) = 0$;
caso contrário, $n(t) = \max\{n(t')\ |\ t'\ \rho\ t\} + 1$.

Definição 3.10. (SelectK-Best) ***SelectK-Best***(K,r) é o operador que retorna o conjunto das K tuplas de r com os menores *níveis*. Caso haja empate entre os níveis das tuplas a serem retornadas, é considerada a ordem em que as tuplas aparecem na relação.

Já o operador ***Select-Best*** seleciona o conjunto das tuplas mais preferidas de uma relação r de acordo com uma teoria-pc Γ. Conforme mencionamos no Capítulo 2, ele utiliza a semântica *"Best Matches Only"* (BMO) ([Kießling 2002]) e é definido da seguinte maneira:

Definição 3.11. (Select-Best) ***Select-Best***$_\Gamma(r) = \{t \in r\ |\ \nexists\ t' \in r,\ t'\ \rho\ t,\ \rho \in \{\succ_\Gamma, >_\Gamma\}\}$

Exemplo 3.7. Considere a instância da relação `Viagens` e a teoria-pc Γ do Exemplo 3.1, descrito no início deste capítulo. De acordo com a ordem forte induzida nesse contexto, mostrada na Figura 3.2, temos: $n(t_1) = n(t_3) = n(t_6) = 0$; $n(t_2) = 1$; $n(t_4) = \max\{n(t_1), n(t_2), n(t_6)\} = 2$; $n(t_5) = \max\{n(t_1), n(t_2), n(t_4), n(t_6)\} = 3$.

Logo,

- ***SelectK-Best***(4,`Viagens`) = $\{t_1, t_3, t_6, t_2\}$.
- ***SelectK-Best***(2,`Viagens`) = $\{t_1, t_3\}$. Note que $n(t_1) = n(t_3) = n(t_6) = 0$, mas t_6 não é considerada na resposta uma vez que t_6 vem depois de t_3 em `Viagens`.
- ***SelectK-Best***(0,`Viagens`) = $\emptyset$.
- ***Select-Best***(`Viagens`) = $\{t_1, t_3, t_6\}$, que é o conjunto das tuplas mais preferidas, aquelas que não são dominadas por nenhuma outra.

3.2.2 Sintaxe CPref-SQL

A proposta da CPref-SQL é que o usuário expresse primeiro suas preferências através do comando `CREATE PREFERENCES`. Para realizar consultas com preferências, basta, então, adicionar a cláusula `ACCORDING TO PREFERENCES` ao bloco do comando `SELECT`, referenciando as preferências criadas.

Comando `CREATE PREFERENCES`

```
CREATE PREFERENCES <nome_preferencia>
FROM <lista_relacoes>
AS <lista_regras-pc>
```

Esse comando cria um conjunto de preferências, denominado `<nome_preferencia>` sobre as relações especificadas em `<lista_relacoes>` e as armazena no catálogo.

Na sentença `<lista_regras-pc>` as preferências são especificadas através de regras-pc. Cada regra é declarada seguindo a sintaxe `IF <antecedente> THEN <consequente> <lista_atributos>`. Várias regras são conectadas pela palavra-chave `and`.

As regras-pc podem ser criadas com ou sem antecedentes, que são termos na forma `atributo predicado valor`, conectados também por `AND`.

Os consequentes são sempre uma relação de preferência entre valores sobre o mesmo atributo, com a sintaxe do tipo: `atributo predicado valor > atributo predicado valor`.

O parâmetro opcional `<lista_atributos>`, que é representado entre colchetes, indica os atributos que são menos importantes ou que não levam em conta a semântica *ceteris paribus*, dependendo da ordem de preferência em questão.

Exemplo 3.8. Para ilustrar, considere que queiramos criar as preferências do usuário sobre as opções de `Viagens`, declaradas no Exemplo 3.1, através da teoria-pc Γ. O comando será o seguinte:

```
CREATE PREFERENCES myprefs
FROM Viagens AS
```

```
P < 3000 > P >= 3000 [D, R, Du] AND
R = 'urbano' > R = 'cruzeiro' [D, Du, C] AND
IF R = 'urbano' THEN C = 'nacional' > C = 'internacional' [D, Du];
```

Vale ressaltar que as preferências expressas pelo comando são armazenadas no catálogo do banco de dados. Para tanto, é feito um mapeamento das regras-pc sobre o esquema formado por `<lista_relacoes>`:

1. cada linha representa uma regra,
2. cada termo da regra do tipo `atributo predicado valor` é transformado no par `(predicado,valor)`,
3. se o atributo for antecedente, o par correspondente a esse atributo é atribuído ao campo,
4. se o atributo for consequente, seus pares são atribuídos ao campo, separados pelo caracter '|',
5. se o atributo estiver entre colchetes é atribuído o valor '*' ao campo,
6. se o atributo não aparece na regra, seu valor é NULL.

A Figura 3.10 ilustra a maneira como as preferências *myprefs* do Exemplo 3.8 serão armazenadas no catálogo.

	DES	ROTEIRO	PREÇO	DUR	CATEGORIA
φ_1	*	*	(<,3000)|(>=,3000)	*	null
φ_2	*	(=,urbano)|(=,cruzeiro)	null	*	*
φ_3	*	(=,urbano)	null	*	(=,nacional)|(=,internacional)

Figura 3.10: Catálogo para as preferências *myprefs*

Cláusula `ACCORDING TO PREFERENCES`

Uma vez criadas as preferências, é possível executar uma consulta CPref-SQL. Um bloco básico de consulta tem a seguinte forma:

```
SELECT <lista_atributos>
FROM <lista_relacoes>
WHERE <condicoes_where>
ACCORDING TO PREFERENCES (<nome_preferencia> [, K])
```

O parâmetro opcional K é um inteiro não-negativo que permite selecionar as K melhores tuplas, invocando o operador ***SelectK-Best***. Se K não for especificado, a consulta será executada com o ***Select-Best*** e retornará apenas as tuplas dominantes.

Exemplo 3.9. Se o usuário que criou suas preferências *myprefs* sobre as opções de viagens quiser realizar a seguinte consulta: *"Desejo uma lista com os 4 pacotes de viagens que mais se adequam às minhas preferências, desde que não sejam opções com roteiro ecológico"*, o comando CPref-SQL correspondente será:

```
SELECT *
FROM Viagens
WHERE roteiro <> 'ecologico'
ACCORDING TO PREFERENCES (myprefs, 4);
```

As consultas com preferências condicionais em CPref-SQL, que contêm a cláusula `ACCORDING TO PREFERENCES`, são denominadas *consultas-pc top-k*.

3.2.3 Plano de Execução de uma Consulta-pc Top-K

O plano canônico de execução associado ao bloco CPref-SQL é mostrado na Figura 3.11. σ, π e $\bowtie$ denotam os usuais operadores da álgebra relacional *seleção*, *projeção* e *junção*, respectivamente. SB e SBK são os operadores da CPref-SQL. Note que a proposta é que os operadores de preferência sejam executados imediatamente após a seleção e antes da projeção.

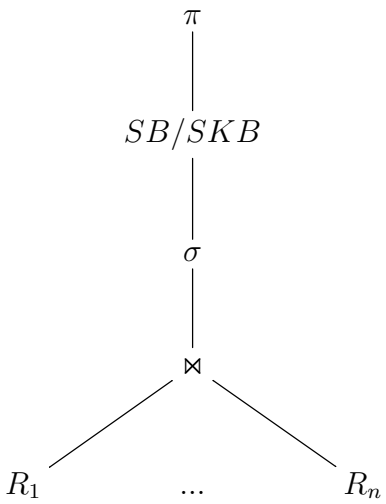


Figura 3.11: Plano canônico de execução de uma consulta CPref-SQL

3.3 Considerações Finais

Neste capítulo a linguagem CPref-SQL foi definida. Primeiramente apresentamos o modelo de preferência adotado pela linguagem: as preferências são expressas através de regras condicionais com predicados genéricos, que induzem uma ordem sobre as tuplas do

banco. Essa ordem de preferência pode ser de dois tipos, forte ou fraca. A diferença é que a primeira é menos expressiva, enquanto a segunda é mais relaxada e consegue comparar mais tuplas. Discutimos o problema da consistência de uma teoria-pc e apresentamos a solução que implementamos para garantir a consistência das teorias submetidas ao banco de dados. Terminamos o capítulo descrevendo os operadores da linguagem, sua sintaxe e o plano canônico de processamento de consultas-pc top-K.

Capítulo 4

Algoritmos e Implementação

Neste capítulo apresentamos os algoritmos propostos e implementações realizadas para a avaliação de consultas CPref-SQL.

Primeiramente, na Seção 4.1, detalhamos os algoritmos que realizam o teste de dominância entre duas tuplas, ou seja, eles comparam as tuplas utilizando as abordagens de ordem forte ou ordem fraca que apresentamos no capítulo anterior. Nas Seções 4.2 e 4.3 apresentamos os algoritmos G-BNL e GRank-BNL, respectivamente, que resolvem os operadores ***Select-Best*** e ***SelectK-Best***.

Em seguida, na Seção 4.4, descrevemos os detalhes da implementação da linguagem CPref-SQL, feita diretamente no código-fonte do SGBDR PostgreSQL: por que escolhemos a abordagem *built-in* para implementação? Quais módulos do SGBDR foram alterados? E se a implementação fosse *on-top*? Essas são algumas das perguntas que respondemos neste tópico.

Por fim, na Seção 4.5, detalhamos como é possível traduzir uma consulta CPref-SQL em linguagem SQL com recursão, mostrando que, de fato, a CPref-SQL não aumenta o poder de expressão da SQL, conforme discutimos nos capítulos anteriores.

Na Seção 4.6 são feitas as considerações finais do capítulo.

4.1 Teste de Dominância

O teste de dominância é um ponto essencial nos algoritmos da CPref-SQL. O problema da dominância pode ser enunciado da seguinte maneira:

Definição 4.1. (Problema da Dominância) Dada uma teoria-pc Γ sobre uma esquema relacional $R = (A_1, ..., A_n)$ e duas tuplas t_1 e t_2 sobre R, decida se $t_1 \ \rho \ t_2$, para $\rho \in \{>_\Gamma, \succ_\Gamma\}$.

Assim, testar a dominância significa decidir se há uma *ordem de preferência* entre duas tuplas. Lembrando que t_1 pode dominar t_2 porque isso é diretamente inferido de uma regra-pc de Γ, ou porque é inferido por transitividade.

Dado que neste trabalho propusemos dois tipos de ordem de preferência que podem ser utilizados com o modelo de preferência da CPref-SQL, a seguir detalhamos os algoritmos para cada uma das abordagens: ordem forte e ordem fraca.

4.1.1 Ordem Forte

O algoritmo *strongDomTest* sintetiza o teste de dominância sob a abordagem de ordem forte. Basicamente, ele utiliza um programa Datalog D_Γ, referente à teoria-pc Γ em questão, para comparar duas tuplas t_1 e t_2. Como retorno, ele informa se existe uma ordem forte de preferência induzida sobre t_1 e t_2 e, se sim, qual é essa ordem. O esquema da Figura 4.1 ilustra a ideia geral do algoritmo.

Figura 4.1: Esquema do algoritmo **strongDomTest**

Antes de descrevermos o algoritmo *strongDomTest*, detalhamos como funciona a ideia de um programa Datalog para resolver o problema da dominância.

Um programa D_Γ, referente à teoria-pc Γ, é construído da seguinte maneira:

(1) para cada regra-pc $r \in \Gamma$ do tipo: $P_{i_1}(A_{i_1}) \wedge ... \wedge P_{i_k}(A_{i_k}) \rightarrow Q_{1_j}(A_j) > Q_{2_j}(A_j)$ $[A_{j_1}, ..., A_{j_m}]$, associamos uma cláusula de Horn p_r:

$$\begin{aligned} pref(x_1, ..., x_n, y_1, ..., y_n) \leftarrow\ & x_{i_1} \models P_{i_1}(A_{i_1}), y_{i_1} = x_{i_1},\ ...,\ x_{i_k} \models P_{i_k}(A_{i_k}), y_{i_k} = x_{i_k}, \\ & x_j \models Q_{1_j}(A_j),\ y_j \models Q_{2_j}(A_j), \\ & x_{k_1} = y_{k_1}, ..., x_{k_l} = y_{k_l}. \end{aligned}$$

onde $j \notin \{i_1, ..., i_k\}$, os conjuntos $\{j, i_1, ..., i_k\}$, $\{j_1, ..., j_m\}$, $\{k_1, ..., k_l\}$ são disjuntos e a união deles é $\{1, ..., n\}$.

(2) consideramos mais duas últimas cláusulas, responsáveis pelo fecho transitivo:

$$\begin{aligned} dom(x_1, ..., x_n, y_1, ..., y_n) &\leftarrow pref(x_1, ..., x_n, y_1, ..., y_n). \\ dom(x_1, ..., x_n, y_1, ..., y_n) &\leftarrow pref(x_1, ..., x_n, z_1, ..., z_n), \\ & \quad dom(z_1, ..., z_n, y_1, ..., y_n). \end{aligned}$$

O teste "$t_1 = (a_1, ..., a_n) > t_2 = (b_1, ..., b_n)$?" é obtido pela execução do *goal* $\leftarrow$ $dom(a_1, ..., a_n, b_1, ..., b_n)$.

Para simplificar o teste de dominância, antes da construção de D_Γ é possível fazer uma análise sobre os atributos que aparecem em Γ para certificar se seus valores precisam

necessariamente ser analisados. É como uma redução de *overhead* no programa, feita da seguinte maneira: seja $\Gamma = \{\varphi_1, ..., \varphi_p\}$, e para cada $j \in \{1, ..., p\}$ seja W_j o conjunto de atributos entre colchetes de φ_j. Seja $V = Atr(R) - \bigcap_1^p W_j$. Então, apenas os atributos pertencentes a V são considerados para montar os predicados *pref* e *dom* em D_Γ.

Como justificativa, considere $t'_1 = t_1[V]$ e $t'_2 = t_2[V]$, ou seja, t'_1 e t'_2 são projeções sobre o conjunto de atributos V. Então, é fácil perceber que $t_1 >_\Gamma t_2$ se e somente se $t'_1 >_\Gamma t'_2$, uma vez que para cada $j \in \{1, ..., p\}$ os valores dos atributos $A \in W_j$ não são levados em conta durante a comparação entre t_1 e t_2, utilizando a regra-pc φ_j.

Dessa forma, uma vez construído o programa D_Γ, a rotina *strongDomTest*, ilustrada na Figura 4.2, gerencia o teste de dominância da ordem forte. A função *execGoal*(t_1, t_2, D_Γ) testa se $t_1 >_\Gamma t_2$, retornando o resultado do $goal \leftarrow dom(a_1, ..., a_n, b_1, ..., b_n)$ no programa D_Γ.

strongDomTest(t_1, t_2, D_Γ) :

Input: duas tuplas t_1 e t_2 e um programa Datalog D_Γ
Output: 1: se $t_1 >_\Gamma t_2$
0: se t_1 é incomparável com t_2
-1: se $t_2 >_\Gamma t_1$

```
1: if execGoal(t1, t2, DΓ) = true then
2:     return 1
3: if execGoal(t2, t1, DΓ) = true then
4:     return -1
5: return 0
```

Figura 4.2: Algoritmo **strongDomTest**

Exemplo 4.1. Para ilustrar o teste de dominância da ordem forte, consideremos a seguinte teoria-pc Γ sobre uma relação `Viagens(D,R,P,Du,C)`:

$$\Gamma = \left\{ \begin{array}{l} \varphi_1 : Du >= 4 > Du < 4\ [D], \\ \varphi_2 : C = nac \rightarrow R = cruz > R = praia\ [\{D, Du\}], \\ \varphi_3 : P < 4000 \wedge C = internac \rightarrow R = praia > R = urb\ [\{D, Du\}]. \end{array} \right\}$$

Quando uma consulta-pc top-k for realizada, os primeiros passos são referentes à construção do programa D_Γ:

- Redução de *overhead*: de acordo com Γ, temos: $W_1 \cap W_2 \cap W_3 = \{D\} \cap \{D, Du\} \cap \{D, Du\} = \{D\}$. Então, $V = \{R, P, Du, C\}$.
- O programa D_Γ será:

  $pref(r_1, p_1, du_1, c_1, r_2, p_2, du_2, c_2) \leftarrow du_1 \geq 4,\ du_2 \leq 4,\ r_1 = r_2,\ p_1 = p_2,\ c_1 = c_2.$

  $pref(r_1, p_1, du_1, c_1, r_2, p_2, du_2, c_2) \leftarrow c_1 = \text{nac},\ c_2 = c_1,\ r_1 = cruz,\ r_2 = praia,\ p_1 = p_2.$

$$pref(r_1, p_1, du_1, c_1, r_2, p_2, du_2, c_2) \leftarrow p_1{<}4000, p_2{=}p_1, c_1{=}internac, c_2{=}c1, r_1{=}praia, r_2{=}urb.$$

$$dom(r_1, p_1, du_1, c_1, r_2, p_2, du_2, c_2) \leftarrow pref(r_1, p_1, du_1, c_1, r_2, p_2, du_2, c_2).$$

$$dom(r_1, p_1, du_1, c_1, r_2, p_2, du_2, c_2) \leftarrow pref(r_1, p_1, du_1, c_1, r_3, p_3, du_3, c_3), \\ dom(r_3, p_3, du_3, c_3, r_2, p_2, du_2, c_2).$$

Suponha que durante a consulta, queiramos inferir uma ordem entre as tuplas $t_1 = (Honolulu, praia, 3000, 3, internac)$ e $t_2 = (NovaYork, urb, 3000, 5, internac)$. De acordo com o teste de dominância da ordem forte *strongDomTest*, temos:

- São consideradas as projeções $t_1[V]$ = (*praia*, 3000, 3, *internac*) e $t_2[V]$ = (*urb*, 3000, 5, *internac*);
- Pelo passo 1, a rotina *execGoal*(t_1, t_2, D_Γ) executa o *goal* *dom*(*praia*, 3000, 3, *internac*, *urb*, 3000, 5, *internac*). Como ele não é satisfeito, retorna *false*.
- No passo 3, *execGoal*(t_2, t_1, D_Γ) é novamente chamada e executa *dom*(*urb*, 3000, 5, *internac*, *praia*, 3000, 3, *internac*) em D_Γ. Como t_2 também não é preferida a t_1, o *goal* não é satisfeito e retorna *false*.
- Pelo passo 5, *strongDomTest* retorna 0, indicando que t_1 e t_2 são incomparáveis.

A ideia é que D_Γ seja construído uma única vez a cada consulta-pc, com base nas preferências referenciadas na cláusula `ACCORDING TO PREFERENCES`. Assim, para inferir uma ordem de preferência entre um par de tuplas, o teste de dominância *strongDomTest* é invocado, simbolizando a tentativa de satisfazer um *goal* em D_Γ.

Quanto à complexidade, a rotina *execGoal* é $O(m^m)$, onde m é o número de regras da teoria-pc. Ou seja, *strongDomTest* possui complexidade exponencial no número de regras, já que implementamos o cálculo do fecho transitivo através da resolução SLD, em um programa com recursão. Vale lembrar também que essa é a complexidade para o pior caso, no qual as duas tuplas são incomparáveis e não se encaixam em nenhuma regra-pc.

4.1.2 Ordem Fraca

O algoritmo *weakDomTest* é o responsável por testar se existe uma ordem fraca de preferência entre duas tuplas, dada uma teoria-pc Γ. Com base nos conceitos desenvolvidos no Capítulo 3, para calcular a ordem fraca é preciso levar em conta as relações entre os atributos da teoria-pc, apontadas pelo grafo de dependência $G(\Gamma)$. A Figura 4.3 ilustra o esquema de *input* e *output* do algoritmo *weakDomTest*.

Para testar se existe uma ordem de preferência fraca entre duas tuplas, também utilizamos o artifício de programas Datalog. Quando uma consulta-pc é submetida ao SGBDR, consideramos o grafo de dependência $G(\Gamma)$ correspondente à teoria-pc Γ envolvida na

Figura 4.3: Esquema do algoritmo **weakDomTest**

consulta, e a cada nó X de G(Γ), associamos um programa Datalog D_X, construído da seguinte maneira:

(1) para cada regra-pc $r \in \Gamma$ do tipo: $P_{i_1}(A_{i_1}) \wedge ... \wedge P_{i_k}(A_{i_k}) \rightarrow Q_1(X) > Q_2(X)$ $[A_{j_1}, ..., A_{j_m}]$, associamos uma cláusula de Horn p_r:

$$pref(x_1, ..., x_n, y_1, ..., y_n) \leftarrow x_{i_1} \models P_{i_1}(A_{i_1}), y_{i_1} = x_{i_1},\ ...,\ x_{i_k} \models P_{i_k}(A_{i_k}), y_{i_k} = x_{i_k},$$
$$x_j \models Q_1(X),\ y_j \models Q_2(X).$$

onde $j \notin \{i_1, ..., i_k\}$, os conjuntos $\{j, i_1, ..., i_k\}$, $\{j_1, ..., j_m\}$, $\{k_1, ..., k_l\}$ são disjuntos e a união deles é $\{1, ..., n\}$. Note que os atributos entre colchetes de uma regra-pc não são levados em conta na semântica da ordem fraca.

(2) consideramos mais duas últimas cláusulas, responsáveis pelo fecho transitivo:

$$dom(x_1, ..., x_n, y_1, ..., y_n) \leftarrow pref(x_1, ..., x_n, y_1, ..., y_n).$$
$$dom(x_1, ..., x_n, y_1, ..., y_n) \leftarrow pref(x_1, ..., x_n, z_1, ..., z_n),$$
$$dom(z_1, ..., z_n, y_1, ..., y_n).$$

Uma vez construídos os programas D_X associados a cada nó de G(Γ), o teste de dominância de ordem fraca é realizado pela rotina $weakDomTest(t_1, t_2, G(\Gamma))$, cujo algoritmo é descrito na Figura 4.4. Nesse teste, devemos considerar o cálculo dos conjuntos Δ e $inf\Delta$, antes de submeter as tuplas ao Datalog.

A função $execGoal(t_1, t_2, D_X)$ corresponde ao teste $t_1 \succ_u^X t_2$, ou seja, retorna o resultado do $goal \leftarrow dom(a_1, ..., a_n, b_1, ..., b_n)$ no programa D_X. Se não existe nenhum programa associado a X, $execGoal$ retorna *false*.

Exemplo 4.2. Considere a teoria-pc Γ do Exemplo 4.1 sobre a relação `Viagens(D,R,P,Du,C)`. Assim como na ordem forte, quando uma consulta-pc top-k é realizada, o primeiro passo é a construção dos programas Datalog associados aos nós de G(Γ):

- O grafo de dependência com os respectivos programas D_R e D_{Du} é ilustrado pela Figura 4.5.

Suponha que queiramos inferir uma ordem fraca sobre as tuplas $t_1 = (Honolulu, praia, 3000, 3, internac)$ e $t_2 = (NovaYork, urb, 3000, 5, internac)$, como no Exemplo 4.1. De acordo com o algoritmo $weakDomTest$, temos:

weakDomTest$(t_1, t_2, G(\Gamma))$:

Input: duas tuplas t_1 e t_2 e o grafo de dependência $G(\Gamma)$

Output: 1: se $t_1 \succ_\Gamma t_2$
0: se t_1 é incomparável com t_2
-1: se $t_2 \succ_\Gamma t_1$

1: $\Delta = \emptyset$
2: **for all** atributo X das tuplas t_1 e t_2 **do**
3: **if** $t_1[X] \neq t_2[X]$ **then**
4: $\Delta = \Delta \cup \{X\}$
5: $inf\Delta = \Delta$
6: **for all** atributo X e $Y \in inf\Delta$ **do**
7: **if** X é descendente de Y em $G(\Gamma)$ **then**
8: $inf\Delta = inf\Delta - \{X\}$
9: **if** $execGoal(t_1, t_2, D_X) = true$ para todo $X \in inf\Delta$ **then**
10: **return** 1
11: **if** $execGoal(t_2, t_1, D_X) = true$ para todo $X \in inf\Delta$ **then**
12: **return** -1
13: **return** 0

Figura 4.4: Algoritmo **weakDomTest**

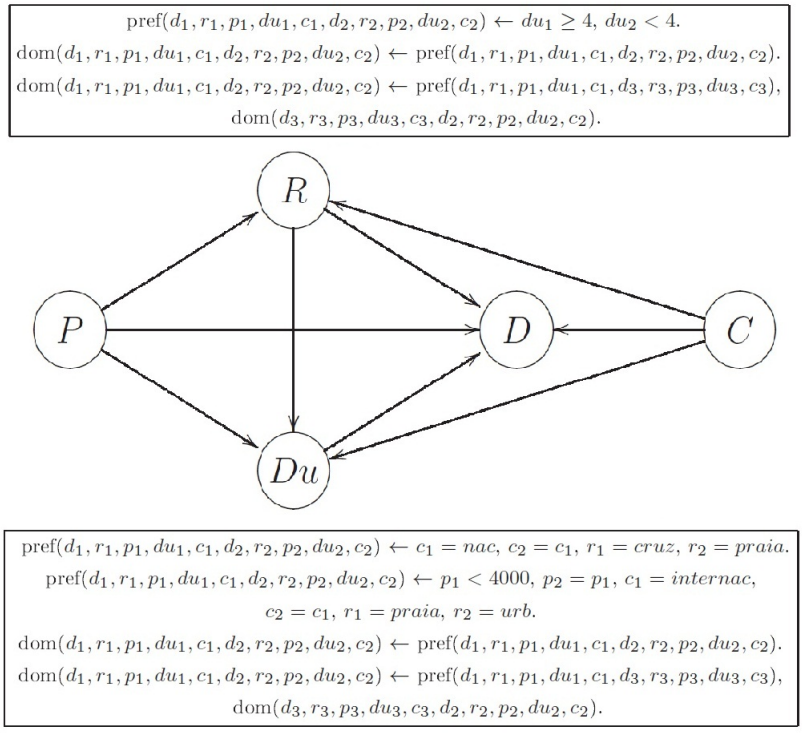


Figura 4.5: G(Γ) com programas Datalog associados aos nós R e Du.

- Pelos passos de 1 a 4, Δ = {D,R,Du} (atributos nos quais t_1 e t_2 se diferenciam);
- $inf\Delta$ = {R} (atributos de Δ que não têm descendentes em Δ), pelos passos de 5 a 8;
- No passo 9, $execGoal(t_1, t_2, D_R)$ é chamada e executa o *goal* $dom(Honolulu, praia, 3000, 3, internac, NovaYork, urb, 3000, 5, internac)$. Como o *goal* é satisfeito, retorna *true*;
- No passo 10, o algoritmo retorna 1, indicando que $t_1 \succ_\Gamma t_2$.

Aqui, novamente a rotina mais onerosa é a *execGoal*, com complexidade $O(m^m)$, onde m é o número de regras associadas ao atributo X (regras cujo atributo X é consequente). Logo, *strongDomTest* também possui complexidade exponencial no número de regras. Entretanto, o cálculo da ordem fraca possui uma constante de multiplicação maior do que a ordem forte, relativa ao tempo de cálculo dos conjuntos Δ e $inf\Delta$.

Fazendo uma análise das complexidades dos algoritmos para teste de dominância, podemos concluir que são bastante onerosos. Entretanto, o teste de dominância está em função da quantidade de regras da teoria-pc envolvida numa consulta. Como as regras são parâmetros informados pelo usuário, a tendência é que as teorias-pc não sejam muito grandes, sendo praticável o tempo exponencial, conforme mostraremos no Capítulo 5, com os resultados experimentais.

4.2 Algoritmo G-BNL

O algoritmo G-BNL (*Generic* BNL) implementa o operador ***Select-Best*** e é descrito na Figura 4.6. A rotina principal **MostPref**(r) retorna as tuplas preferidas da relação r. Vale ressaltar que ele emprega a estratégia de *block nested looping* (laço aninhado em blocos) do algoritmo BNL de [Börzsönyi et al. 2001].

Mais uma vez, a parte essencial do algoritmo é o teste de dominância. A rotina **dominanceTest**(t, t') simboliza uma chamada ao teste de dominância em questão. Ou seja, ela pode ser subentendida como uma chamada ao procedimento **strongDomTest**(t, t', D_Γ) ou ao procedimento **weakDomTest**($t, t', G(\Gamma)$), dependendo da abordagem que o código estiver seguindo. Lembramos que os parâmetros D_Γ e G(Γ) são estruturas montadas apenas uma vez no início da avaliação da consulta-pc e estão subentendidas no contexto da chamada $dominanceTest(t, t')$.

MostPref(r) :
Input: um conjunto r de tuplas
Output: um conjunto S contendo as tuplas mais preferidas de r

```
1: limpe a página de memória W e a tabela temporária F
2: faça r o input
3: while input não é vazio do
4:     for all tupla t do input do
5:         dominate = -1
6:         for all tupla t' ∈ W do
7:             dominanceTest(t, t')
8:             if t é dominada por t' then
9:                 dominate = 0
10:                break
11:            if t domina t' then
12:                remove t' de W
13:                dominate = 1
14:        if dominate = 1 then
15:            insere t em W
16:        else if dominate = -1 then
17:            insere t em W se houver espaço, caso contrário insere t em F
18:    insere em S as tuplas de W que foram adicionadas quando F estava vazio
19:    faça F o novo input, limpe a tabela temporária
20: return S
```

Figura 4.6: Algoritmo **G-BNL**

Exemplo 4.3. Para ilustrar a execução do G-BNL, consideremos que a seguinte consulta-pc foi submetida ao sistema:

```
CREATE PREFERENCES myprefs
FROM Viagens AS
 P < 3000 > P >= 3000 [D, R, Du] AND
 R = 'urbano' > R = 'cruzeiro' [D, Du, C] AND
 IF R = 'urbano' THEN C = 'nacional' > C = 'internacional' [D, Du];

SELECT *
FROM Viagens
ACCORDING TO PREFERENCES (myprefs);
```

Suponha que uma instância r de `Viagens` seja constituída pelas tuplas t_1, t_2, t_3, t_4, t_5 e t_6 e que a **ordem fraca** de preferência induzida sobre r está representada no grafo *better-than* da Figura 4.7 (grafo referente ao contexto do Exemplo 3.5).

Para ilustrar todos os estágios do algoritmo (especialmente quando o *buffer* está cheio), supomos, ainda, que a página de memória W possa armazenar apenas duas tuplas.

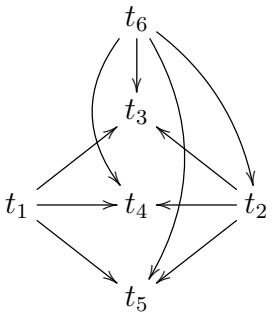


Figura 4.7: Grafo *better-than* da relação `Viagens` sob a ordem fraca de preferência.

Com isso, para avaliar a consulta acima, é feita a chamada **MostPref**(r), que tem o seguinte comportamento:

* Pelo passo 5, *dominate* = -1;
* como W é vazia, no passo 17, insere t_1 em W: $W = \{t_1\}$ (Figura 4.8(a));
* como a análise diz respeito a ordem fraca, no passo 7, é chamada $weakDomTest(t_2, t_1, G(\Gamma))$;
* t_2 e t_1 são incomparáveis. Pelo passo 17, insere t_2 em W: $W = \{t_1, t_2\}$;
* pelo passo 7, $weakDomTest(t_3, t_1, G(\Gamma))$ = -1. Nos passos de 8-10, *dominate* = 0 e ignora t_3;
* No passo 5, *dominate* = -1;
* $weakDomTest(t_4, t_1, G(\Gamma))$ = -1, pelo passo 7. Nos passos de 8-10, *dominate* = 0 e ignora t_4;
* No passo 5, *dominate* = -1;
* No passo 7, $weakDomTest(t_5, t_1, G(\Gamma))$ = -1. *dominate* = 0 e ignora t_5, pelos passos de 8-10 (Figura 4.8 (b));
* No passo 5, *dominate* = -1;
* $weakDomTest(t_6, t_1, G(\Gamma)) = 0$, pelo passo 7;
* Novamente, no passo 7, $weakDomTest(t_6, t_2, G(\Gamma)) = 1$;
* Pelos passos de 11-13: remove t_2 de W e *dominate* = 1;
* No passo 15, insere t_6 em W: $W = \{t_1, t_6\}$;
* $S = \{t_1, t_6\}$, pelo passo 18 (Figura 4.8(c));
* F é o novo input no passo 19;
* Como F é vazio, a condição do passo 3 falha;
* Retorna $S = \{t_1, t_6\}$.

A complexidade do G-BNL é $O(n^2(m^m))$, onde n é a cardinalidade da relação de *input* r e m é o número de regras da teoria-pc. $O(m^m)$ corresponde ao tempo do teste de dominância. Como $n >> m$, então sua complexidade tende a $O(n^2)$.

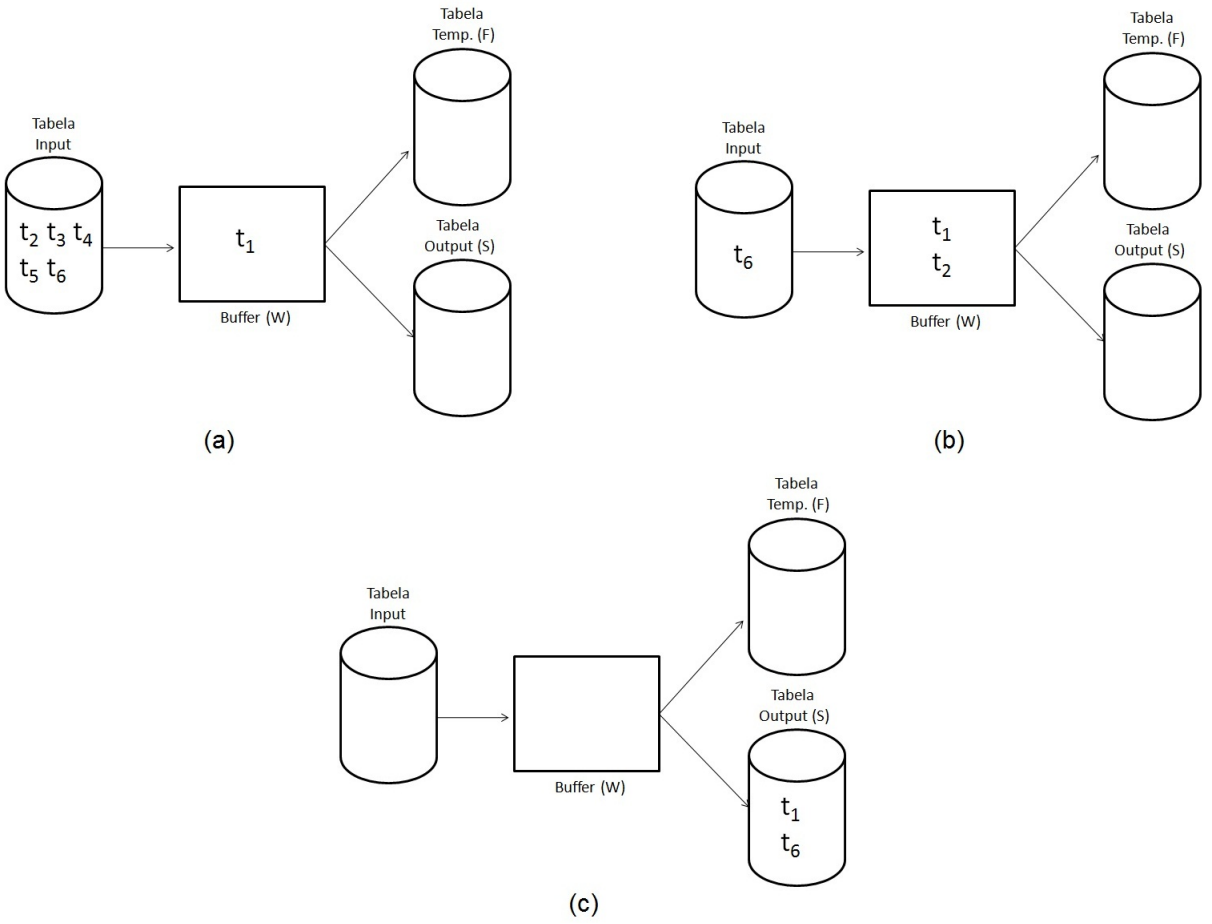


Figura 4.8: Iterações do algoritmo G-BNL

4.3 Algoritmo GRank-BNL

O operador ***SelectK-Best*** é implementado pelo algoritmo GRank-BNL, através da rotina **Top**(r, K), onde r é um conjunto de tuplas e K é um inteiro não-negativo. O algoritmo está descrito na Figura 4.9. Assim como o G-BNL, o GRank-BNL também segue as linhas dos algoritmos de laços aninhados.

Conforme a definição do operador ***SelectK-Best***, o objetivo agora é obter os *níveis* de cada tupla e só depois selecionar aquelas com os menores níveis. Note que, no passo 6, a rotina **topK**(r), descrita na Figura 4.10, é chamada. Ela é responsável por avaliar a lista MorePref(t_i) de cada tupla t_i no *input* r. MorePref(t_i) é a lista de todas as tuplas $t_j \in r$ que dominam t_i, ou seja, MorePref(t_i) = $\{t_j \in r \mid t_j$ é preferida a $t_i\}$. Depois de obter as listas MorePref(t_i) para cada $t_i \in r$, o segundo passo do algoritmo percorre r para obter as K tuplas com os menores níveis. Da Definição 3.9, lembramos que o nível de uma tupla t_i é dado por:

$$\text{n}(t_i) = 0, \text{ para MorePref}(t_i) = \emptyset$$
$$\text{n}(t_i) = \max(\{n(t_j) \mid t_j \in MorePref(t_i)\}) + 1, \text{ para MorePref} \neq \emptyset.$$

Novamente, **dominanceTest**(t, t') corresponde a um dos testes de dominância descritos na primeira seção deste capítulo. A utilização dos algoritmos *strongDomTest* ou

Top(r, K) :

Input: um conjunto r de tuplas, um número $K \geq -1$
Output: Para $K = -1$: um conjunto S das tuplas mais preferida de r
Para $K \geq 0$: um conjunto S contendo as K tuplas *"menos dominadas"* de r

```
1:  if K = −1 then
2:      S = MostPref (r)
3:  else if K = 0 then
4:      S = ∅
5:  else if K > 0 then
6:      S′ = topK(r)
7:      S = ∅
8:      currentLevel = 0
9:      while |S| < K and |S| < |S′| do
10:         for toda tupla t ∈ S′ do
11:             if n(t)=currentLevel then // n(t)=max({n(s) | s ∈ MorePref(t)}) + 1
12:                 S = S ∪ {t}
13:         currentLevel = currentLevel + 1
14: return S
```

Figura 4.9: Algoritmo **GRank-BNL**

weakDomTest para resolver o teste de dominância depende da ordem de preferência em questão.

topK(r) :

```
1:  limpe a página de memória W e a tabela temporária F
2:  faça r o input
3:  while input não é vazio do
4:      for all tupla t no input do
5:          for all tupla t′ ∈ W do
6:              dominanceTest(t, t′)
7:              if t é dominado por t′ then adiciona t′ em MorePref(t)
8:              if t domina t′ then adiciona t em MorePref(t′)
9:          insere t em W se há espaço, caso contrário adiciona t em F
10:     insere em S as tuplas de W
11:     faça F o input, limpe a tabela temporária
12: return S
```

Figura 4.10: Algoritmo **topK**

Exemplo 4.4. Suponha agora, que sobre as mesmas preferências *myprefs* do Exemplo 4.3, a seguinte consulta-pc top-k seja submetida ao SGBDR:

```
SELECT *
FROM Viagens
```

```
ACCORDING TO PREFERENCES (myprefs, 4);
```

Considerando que uma instância r de `Viagens` seja formada pelas tuplas t_1, t_2, t_3, t_4, t_5 e t_6, cuja **ordem forte** induzida sobre elas é mostrada pelo grafo da Figura 4.11, a chamada **Top**(r, 4) será acionada e, no passo 6, **topK**(r) é chamada. Considerando também que a janela de memória W tenha espaço para duas tuplas, *topK* terá o seguinte comportamento:

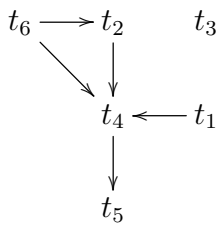


Figura 4.11: Grafo *better-than* da relação `Viagens` sob a ordem forte de preferência.

* Como W está vazia inicialmente, no passo 9, t_1 é inserida em W: $W = \{t_1\}$;
* No passo 6, $strongDomTest(t_2, t_1, D_\Gamma) = 0$, ou seja, t_1 e t_2 são incomparáveis;
* Pelo passo 9, $W = \{t_1, t_2\}$ (Figura 4.12(a));
* $strongDomTest(t_3, t_1, D_\Gamma) = 0$, no passo 6;
* $strongDomTest(t_3, t_2, D_\Gamma) = 0$, no passo 6;
* No passo 9, como não há espaço em W, insere t_3 em F: $F = \{t_3\}$;
* No passo 6, $strongDomTest(t_4, t_1, D_\Gamma)$ = -1;
* Pelo passo 7, MorePref(t_4) = $\{t_1\}$;
* $strongDomTest(t_4, t_2, D_\Gamma)$ = -1, no passo 6;
* Pelo passo 7, MorePref(t_4) = $\{t_1, t_2\}$;
* No passo 9, como não há espaço em W, insere t_4 em F: $F = \{t_3, t_4\}$ (Figura 4.12(b));
* $strongDomTest(t_5, t_1, D_\Gamma)$ = -1, no passo 6;
* Pelo passo 7, MorePref(t_5) = $\{t_1\}$;
* $strongDomTest(t_5, t_2, D_\Gamma)$ = -1, no passo 6;
* Pelo passo 7, MorePref(t_5) = $\{t_1, t_2\}$;
* No passo 9, como não há espaço em W, insere t_5 em F: $F = \{t_3, t_4, t_5\}$;
* $strongDomTest(t_6, t_1, D_\Gamma) = 0$, no passo 6;
* $strongDomTest(t_6, t_2, D_\Gamma) = 1$, no passo 6;
* Pelo passo 8, MorePref(t_2) = $\{t_6\}$;
* No passo 9, como não há espaço em W, insere t_6 em F: $F = \{t_3, t_4, t_5, t_6\}$;
* Pelo passo 10, $S = \{t_1, t_2\}$;
* o novo input é $\{t_3, t_4, t_5, t_6\}$ e $F = W = \emptyset$;

* Como W está vazia inicialmente, no passo 9, t_3 é inserida em W: $W = \{t_3\}$;
* No passo 6, $strongDomTest(t_4, t_3, D_\Gamma) = 0$, ou seja, t_4 e t_3 são incomparáveis;
* Pelo passo 9, $W = \{t_3, t_4\}$;
* $strongDomTest(t_5, t_3, D_\Gamma) = 0$, no passo 6;
* $strongDomTest(t_5, t_4, D_\Gamma)$ = -1, no passo 6;
* Pelo passo 7, MorePref(t_5) = $\{t_4\}$;
* No passo 9, como não há espaço em W, insere t_5 em F: $F = \{t_5\}$;
* $strongDomTest(t_6, t_3, D_\Gamma) = 0$, no passo 6;
* $strongDomTest(t_6, t_4, D_\Gamma) = 1$, no passo 6;
* Pelo passo 8, MorePref(t_4) = $\{t_6\}$;
* No passo 9, como não há espaço em W, insere t_6 em F: $F = \{t_5, t_6\}$ (Figura 4.12(c));
* Pelo passo 10, $S = \{t_1, t_2, t_3, t_4\}$;
* o novo input é $\{t_5, t_6\}$ e $F = W = \emptyset$;
* Como W está vazia inicialmente, no passo 9, t_5 é inserida em W: $W = \{t_5\}$;
* $strongDomTest(t_6, t_5, D_\Gamma) = 1$, no passo 6;
* Pelo passo 8, MorePref(t_5) = $\{t_6\}$;
* Pelo passo 9, $W = \{t_5, t_6\}$;
* Pelo passo 10, $S = \{t_1, t_2, t_3, t_4, t_5, t_6\}$;
* Como $F = \emptyset$, o novo input é vazio, pelo passo 11, e a condição do passo 3 falha;
* Retorna $S = \{t_1, t_2, t_3, t_4, t_5, t_6\}$.

Os próximos passos da execução de **Top**(r, 4) são ilustrados abaixo.

* Nos passos de 6-8, temos: $S' = \{t_1, t_2, t_3, t_4, t_5, t_6\}$, $S = \emptyset$ e $currentLevel = 0$;
* nível de t_1 é computado pelo passo 11: n(t_1) = 0 = $currentLevel$ e $S = \{t_1\}$, pelo passo 12;
* Pelo passo 11, n(t_2) é computado: n(t_2) = max(n(t_6)) + 1 = 1. Então, n(t_2) $\neq$ $currentLevel$;
* nível de t_3 = 0 = $currentLevel$, pelo passo 11. $S = \{t_1, t_3\}$, pelo passo 12;
* Pelo passo 11, n(t_4) é computado: n(t_4) = max(n(t_1), n(t_2), n(t_6)) + 1 = 2. Então, n(t_4) $\neq$ $currentLevel$;
* Pelo passo 11, n(t_5) é computado: n(t_5) = max(n(t_1), n(t_2), n(t_4), n(t_6)) + 1 = 3. Então, n(t_5) $\neq$ $currentLevel$;
* nível de t_6 = 0 = $currentLevel$, pelo passo 11. $S = \{t_1, t_3, t_6\}$, pelo passo 12;
* $currentLevel$ = 1 no passo 13. A execução retorna ao passo 9;
* No passo 11, n(t_2) = 1 = $currentLevel$. Então, $S = \{t_1, t_3, t_6, t_2\}$, pelo passo 12;
* Como $|S| = K = 4$, pelo passo 9, a execução para e retorna $S = \{t_1, t_3, t_6, t_2\}$.

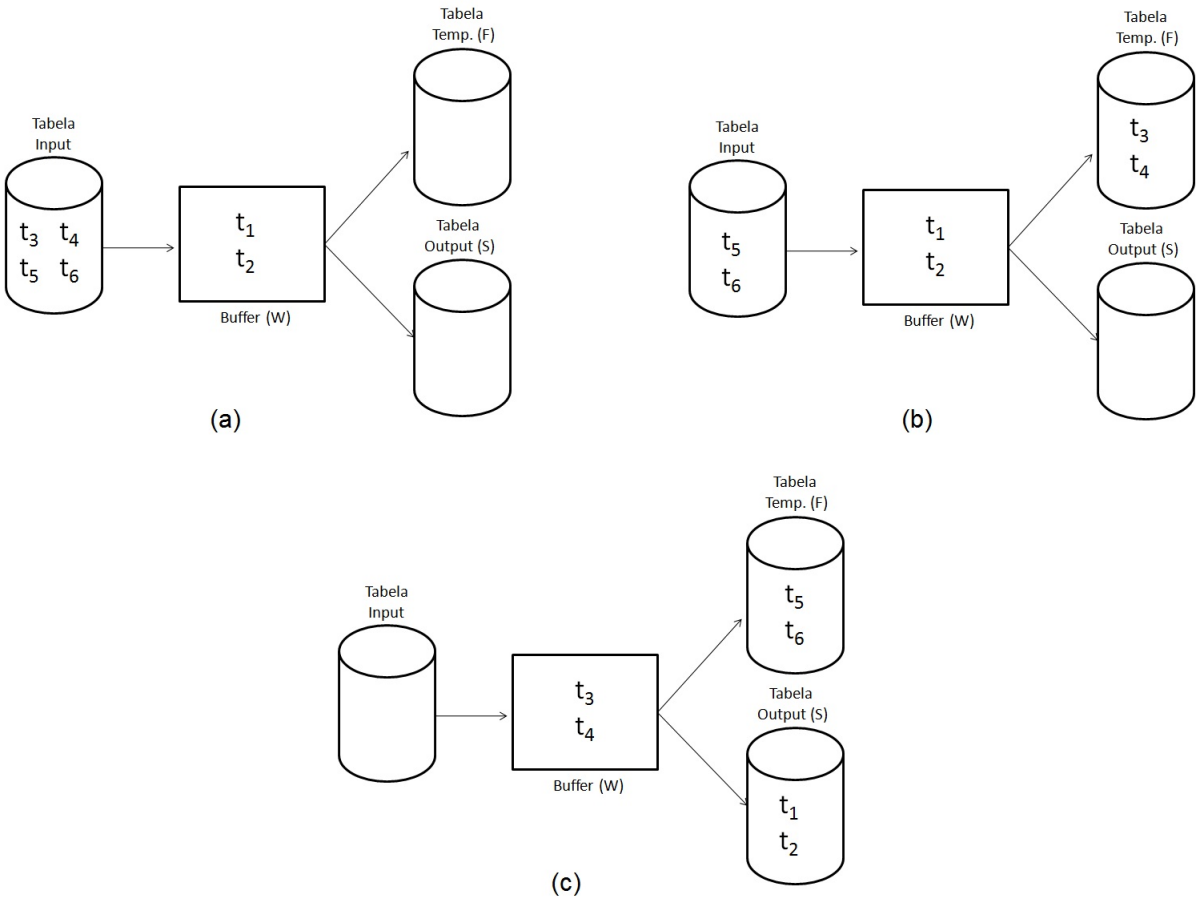


Figura 4.12: Iterações do algoritmo GRank-BNL

Em termos de complexidade, a rotina **topK**, a mais onerosa do algoritmo GRank-BNL, também executa em $O(n^2(m^m))$, onde n é o tamanho da relação r de entrada e m é a quantidade de regras da teoria-pc. O tempo $O(m^m)$ corresponde ao custo do teste de dominância. Como $n >> m$, temos que essa complexidade tende a $O(n^2)$. Entretanto, o tempo total do GRank-BNL possui uma constante de multiplicação maior que o G-BNL, devido o cálculo dos níveis na rotina **Top**.

4.4 Implementação

A implementação da linguagem CPref-SQL foi feita diretamente no código-fonte do SGBDR PostgreSQL 8.4. Essa abordagem *built-in* para codificação foi escolhida por possuir diversas vantagens. Primeiro, pelo fato de podermos inserir os operadores CPref-SQL dentro do plano de execução de consultas, para serem executados antes da projeção, conforme descrevemos no Capítulo 3, Figura 3.11. Além disso, questões como controle de concorrência e gerenciamento de transações não precisam ser tratadas, pois já estão implementadas no próprio gerenciador. Segundo, vislumbramos que se diversos algoritmos

que avaliem os operadores da CPref-SQL estiverem implementados no SGBDR, o sistema é capaz de selecionar o mais eficiente a cada consulta. E terceiro, por ser a abordagem com melhor desempenho, conforme comprovamos nos diversos experimentos realizados. A escolha da implementação sobre o PostgreSQL deve-se ao fato de ser um SGBDR de código aberto.

Nesta seção, apresentamos detalhes de como foi feita a codificação da linguagem na arquitetura PostgreSQL, ressaltando quais módulos foram modificados e como inserimos os algoritmos G-BNL e GRank-BNL no código. Além disso, destacamos as alterações em termos de sintaxe (*parser*), necessárias diante do novo comando `CREATE PREFERENCES` e da cláusula `ACCORDING TO PREFERENCES`.

Discutimos também, como a CPref-SQL poderia ser implementada utilizando a abordagem *on-top*, que é independente do código-fonte do SGBDR. Até que ponto tal abordagem pode ser interessante para o desenvolvimento de um nova linguagem de preferências? Quais as vantagens e desvantagens em relação à implementação *built-in* que escolhemos?

Por fim, diante de diferentes ordens de preferência, abordagens de codificação e ferramentas auxiliares desenvolvidas, apresentamos um resumo de tudo o que foi codificado e como organizamos a disponibilização dos códigos.

4.4.1 CPref-SQL na Arquitetura PostgreSQL

De acordo com [Conway and Sherry 2006], a parte interna do PostgreSQL (*backend*) é formanda por cinco componentes principais. A Figura 4.13 mostra como esses componentes funcionam em conjunto. Os módulos em destaque indicam aqueles que modificamos durante a implementação da CPref-SQL.

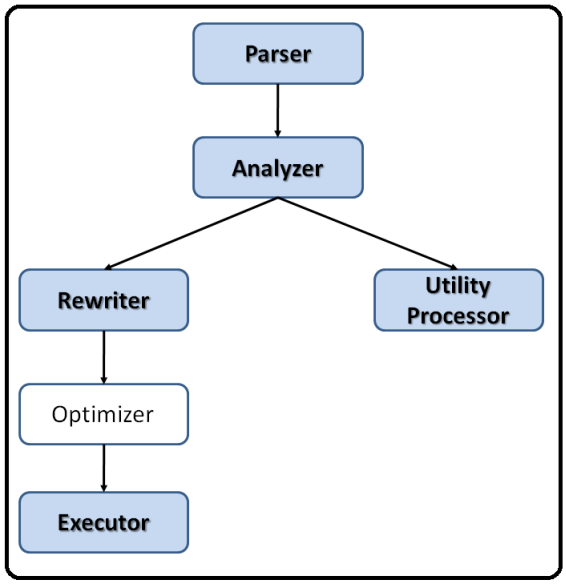


Figura 4.13: Diagrama da arquitetura PostgreSQL. Componentes em azul indicam aqueles modificados para a extensão CPref-SQL

Para entendermos como os algoritmos da CPref-SQL foram inseridos no *backend* do SGBDR, apresentamos uma visão geral desses componentes principais, indicando o que foi acrescentado em cada um.

1. ***parser/analyzer:*** faz a análise léxica de uma *string* de consulta submetida pelo usuário. Estendemos esse módulo com a inserção das novas palavras-chave: `ACCORDING` e `PREFERENCES`. É aqui também que definimos a gramática que expressa a sintaxe do comando `CREATE PREFERENCES` e da cláusula de preferência `ACCORDING TO PREFERENCES`. Para complementar as descrições da Subseção 3.2.2 do capítulo anterior, os diagramas de sintaxe da Figura 4.14, esquematizam como as cláusulas *create_pref_command* e *preference_clause* foram modeladas no arquivo `src/backend/parser/gram.y`[1]. A cláusula de preferência foi basicamente inserida no comando `SELECT`. Já o comando para criação de preferências envolve toda a sintaxe das regras-pc.

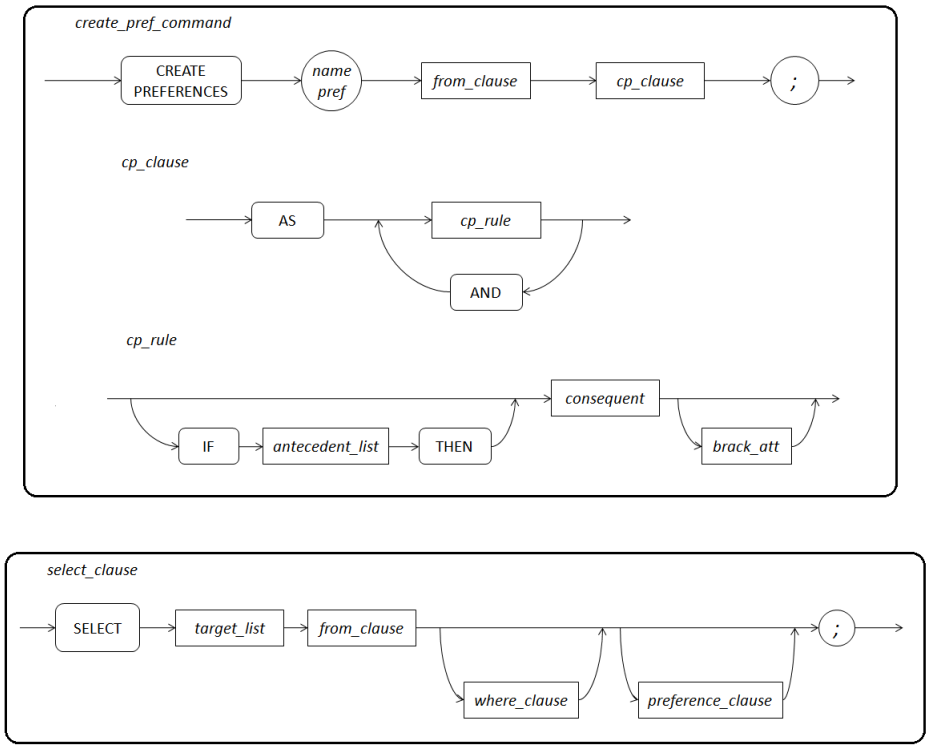


Figura 4.14: Diagramas de sintaxe dos novos comandos CPref-SQL

2. ***rewriter:*** a fase de análise produz uma consulta através da chamada árvore de análise de consulta. Neste módulo, regras de reescrita transformam uma consulta em um plano que encapsula como tal consulta deve ser executada. Montamos, então, o plano de consulta canônico da CPref-SQL, inserindo a chamada aos algoritmos de preferência imediatamente antes da operação de projeção. Os códigos estão no diretório `src/backend/rewrite/`[1].

[1] http://www.postgresql.org/docs/8.4/static/

3. ***optimizer:*** encontra um plano de consulta eficiente. Neste trabalho, não consideramos questões de otimização de consultas, por isso nada foi acrescentado neste módulo.

4. ***executor:*** parte responsável pela execução de uma consulta. Aqui, inserimos os algoritmos que descrevemos neste capítulo. O ponto de partida está no arquivo `src/backend/executor/execMain.c`[2].

5. ***utility processor:*** processa sentenças DDL (*data definition language*), como por exemplo `CREATE TABLE`. Como o comando para criação de preferências é uma sentença DDL, é nesse módulo que codificamos a nova função utilitária `CREATE PREFERENCES` que armazena as preferências de um usuário no catálogo do SGBDR, conforme ilustramos também na Subseção 3.2.2. Os novos arquivos que implementam a lógica do comando foram inseridos no diretório `src/backend/commands/`[2].

4.4.2 Abordagem *On-Top*

Implementar o método de preferência como um programa *stand-alone* ou através de funções definidas pelo usuário no SGBDR é uma alternativa possível em tratando-se da linguagem CPref-SQL. De fato, fazer com que o SGBDR seja tratado como uma caixa-preta e os operadores de preferência totalmente desacoplados das operações internas do banco (junções e seleções, por exemplo), tornam a implementação da linguagem mais simples e flexível, sendo possível manipular preferências sobre diferentes SGBDRs, em suas diferentes versões. Entretanto, os operadores de preferência não podem mais interagir com outros operadores relacionais, e funcionalidades como controle de concorrência não podem ser aproveitadas.

Como uma prova de conceito, implementamos uma versão simplificada da CPref-SQL sob essa abordagem *on-top*. Utilizamos a estratégia de estender o PostgreSQL através de bibliotecas dinâmicas, acionadas por meio de funções do SGBDR. Nessa implementação, codificamos a linguagem para o modelo de preferência simplificado, que utiliza apenas o operador de igualdade nas regras-pc, com a ordem de preferência forte.

A seguir, apresentamos detalhes de implementação e do novo plano de consultas que uma consulta-pc top-k gera. No Capítulo 5 mostraremos uma série de resultados experimentais que ilustram o desempenho dessa versão *on-top* da linguagem.

Detalhes de Implementação

Basicamente, três funções foram criadas: *createPref*, *mostPref* e *topK*.

createPref é a função responsável por criar as preferências do usuário, armazenando-as no banco de dados. Possui a seguinte sintaxe:

[2]http://www.postgresql.org/docs/8.4/static/

```
createPref(<nome_preferencia>, <lista_relacoes>, <lista_regras-pc>)
```

Similar ao comando `CREATE PREFERENCES`, esta função cria um conjunto de preferências, denominado `nome_preferencia` sobre as relações especificadas em `lista_relacoes` e as armazena no catálogo. No parâmetro `lista_regras-pc` as preferências são especificadas através de regras-pc. Cada regra é declarada seguindo a mesma sintaxe do comando `CREATE PREFERENCES: IF <antecedente> THEN <consequente> <lista_atributos>` (para maiores detalhes, vide Subseção 3.2.2).

Exemplo 4.5. Suponha que para expressar suas preferências sobre uma relação `Viagens(D,R,P)`, sob a abordagem *built-in*, o usuário submeta o seguinte comando:

```
CREATE PREFERENCES myprefs
   FROM Viagens AS
    P < 3000 > P >= 3000 [D] AND
    R = 'urbano' > R = 'cruzeiro' [D];
```

O comando correspondente na abordagem *on-top* é:

```
SELECT createPref('myprefs',
   'Viagens',
    'P < 3000 > P >= 3000 [D] AND
    R = ''urbano'' > R = ''cruzeiro'' [D]');
```

mostPref executa uma consulta com preferências que utiliza o operador ***Select-Best***. Possui a seguinte sintaxe:

```
mostPref( <nome_pref>, <consulta> )
```

Quando chamada, *mostPref* executa a consulta especificada em `<consulta>` e, sobre o resultado, aplica o operador ***Select-Best*** utilizando as preferências `<nome_pref>`. O exemplo a seguir ilustra como essa função é executada.

Exemplo 4.6. Para submeter a seguinte consulta-pc:

```
SELECT D, P
   FROM Viagens
```

```
WHERE R <> 'ecologico'
ACCORDING TO PREFERENCES (myprefs);
```

sob a abordagem *on-top*, a chamada ficará assim:

```
SELECT D, R
   FROM mostPref( 'myprefs',
   'SELECT * FROM Viagens
   WHERE R <> ''ecologico''' ) as (D text, R text);
```

topK é a função correspondente à execução do operador ***SelectK-Best***. A única diferença para a função *mostPref* é o parâmetro correspondente ao valor K.

```
topK( <nome_pref>, <K>, <consulta> )
```

Exemplo 4.7. Suponha que desejemos executar a consulta do Exemplo 4.6 para obter exatamente as 4 tuplas mais preferidas, sob a abordagem *on-top*. Então, temos:

```
SELECT D, R
   FROM mostPref( 'myprefs', 4,
   'SELECT * FROM Viagens
   WHERE R <> ''ecologico''' ) as (D text, R text);
```

Note que as projeções feitas nas consultas dentro das chamadas das funções *mostPref* e *topK* são todas com o parâmetro '*', ou seja, sobre todos os atributos das relações envolvidas. A projeção sobre os atributos desejados deve ser feita de fora da chamada da função. Então, é como se sempre duas consultas fossem realizadas: a primeira para calcular as preferências através das funções e a segunda para obter as projeções desejadas sobre os resultados das funções. A Figura 4.15 ilustra o plano canônico de execução de consultas diante dessa implementação.

Outro ponto positivo dessa implementação é que não precisamos limitar a CPref-SQL ao bloco `SELECT ... FROM ... WHERE ...`. Para executar consultas mais elaboradas, basta adequarmos a chamada de função à consulta. Por exemplo, uma consulta do tipo:

```
SELECT D, R, P
FROM mostPref( 'myprefs', 4,
'SELECT * FROM Viagens
```

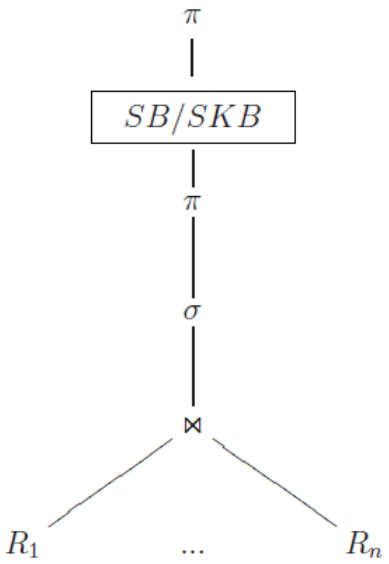


Figura 4.15: Plano canônico de execução de uma consulta CPref-SQL *on-top*

```
WHERE R <> ‘‘ecologico’’’ ) as (D text, R text, P integer) ORDER BY P;
```

pode perfeitamente ser executada. Logo, o desenvolvimento da linguagem CPref-SQL *on-top* é mais rápido, uma vez que, ainda deixamos como trabalho futuro a execução de consultas mais elaboradas sob a abordagem *built-in* por questões de tempo de implementação.

4.4.3 Organização do Código

Disponibilizamos todos os códigos desenvolvidos neste trabalho no endereço: `http://www.lsi.ufu.br/cprefsql`. Neste endereço, também colocamos uma interface de demonstração, na qual podem ser executadas consultas em linguagem CPref-SQL. No Apêndice A, apresentamos uma descrição detalhada desse *site* que desenvolvemos para documentação deste trabalho. Os itens disponíveis para *download* são:

- Código-fonte do PostgreSQL estendido para CPref-SQL sob a ordem de preferência forte;
- Código-fonte do PostgreSQL estendido para CPref-SQL sob a ordem de preferência fraca;
- Ferramenta para teste de consistência;
- Código-fonte e bibliotecas das funções *on-top* que implementam a versão simplificada do modelo de preferência da CPref-SQL;
- Documentações específicas para instalação e execução das ferramentas.

4.5 Reescrita em SQL

Depois de apresentar toda a cadeia de algoritmos e implementação da CPref-SQL, finalizamos este capítulo com uma alternativa para avaliação de consultas com preferências condicionais: através da tradução de consultas CPref-SQL em consultas na linguagem SQL com recursão.

Conforme comentamos no Capítulo 2, os novos operadores não aumentam o poder de expressão da SQL, uma vez que a semântica deles pode ser obtida através de consultas SQL. Na verdade, qualquer operador projetado para produzir as tuplas não dominadas pode ser expresso em SQL [Chomicki 2003].

Entretanto, diferentemente das consultas *Skyline* e Pareto, que podem ser representadas em linguagem SQL-92 padrão [ISO 1992], nossas consultas-pc vão além do poder de expressão da álgebra relacional clássica, envolvendo necessariamente recursão em sua especificação. Isto se deve à operação de fecho transitivo envolvida na definição da ordem de preferência associada a uma teoria-pc. Mostramos, então, como transformar nossas consultas CPref-SQL em consultas SQL:99 [ISO 1999], que é a especificação SQL que suporta recursão com a cláusula `WITH RECURSIVE`.

Dado um conjunto de preferências especificadas através do comando `CREATE PREFERENCES`, cada regra-pc é traduzida num bloco `SELECT ... FROM ... WHERE ...`. O objetivo é combinar em pares as tuplas que satisfazem tal bloco, de maneira que a primeira tupla sempre domina a segunda tupla do par, em relação às condições da regra em questão. No fim, selecionamos recursivamente todas as tuplas que não são dominadas por nenhuma outra.

Exemplo 4.8. Para exemplificar, o comando SQL correspondente à consulta-pc do Exemplo 4.3 é descrito abaixo:

```
CREATE OR REPLACE VIEW Rules (D, R, P, Du, C, des, rot, pre, dur, cat) AS
  (SELECT * FROM Viagens V, Viagens V1
   WHERE V.P<3000 and V1.P≥3000 and V.C=V1.C)
     UNION
  (SELECT * FROM Viagens V, Viagens V1
   WHERE V.R='urbano' and V1.R='cruzeiro' and V.P=V1.P)
     UNION
  (SELECT * FROM Viagens V, Viagens V1
   WHERE V.R='urbano' and V1.R=V.R and V.C='nacional' and
         V1.C='internacional' and V.P=V1.P);

WITH RECURSIVE Recursion (des, rot, pre, dur, cat, D, R, P, Du, C) AS (
  (SELECT * FROM Rules)
     UNION
```

```
(SELECT V.D, V.R, V.P, V.Du, V.C, R.D, R.R, R.P, R.Du, R.C
 FROM Rules V, Recursion R
 WHERE V.des=R.des, V.rot=R.rot, V.pre=R.pre, V.dur=R.dur, V.cat=R.cat))

 SELECT * FROM Viagens
   EXCEPT
SELECT R.D, R.R, R.P, R.Du, R.C FROM Recursion R;
```

Vale ressaltar que desenvolvemos apenas a tradução de consultas com o operador ***Select-Best***, sob a ordem de preferência forte. Note que uma grande vantagem da existência da linguagem CPref-SQL é a expressividade de consultas com preferências. Expressar preferências em SQL não é intuitivo, e acabou tornando-se bastante complexo. No Capítulo 5, apresentaremos uma bateria de testes que comparam o desempenho dessa abordagem de reescrita com a implementação *built-in* da CPref-SQL que propusemos neste trabalho, evidenciando uma segunda vantagem da CPref-SQL: o desempenho.

4.6 Considerações Finais

Neste capítulo, descrevemos os algoritmos propostos para a avaliação das consultas em CPref-SQL. Apresentamos os algoritmos *strongDomTest* e *weakDomTest* que realizam o teste de dominância entre duas tuplas sob a ordem forte e ordem fraca de preferência, respectivamente. Ambos utilizam programas Datalog para calcular o fecho transitivo das ordens de preferência. Também propusemos os algoritmos G-BNL e GRank-BNL que implementam os operadores ***Select-Best*** e ***SelectK-Best***, respectivamente.

Apresentamos uma descrição detalhada acerca da codificação dos algoritmos no *backend* do PostgreSQL, caracterizando a abordagem *built-in* de implementação. Descrevemos, também, a implementação alternativa que desenvolvemos de um protótipo da linguagem CPref-SQL, sob a abordagem *on-top*.

Ao final, mostramos uma proposta para tradução das consultas CPref-SQL em consultas SQL com recursão.

Capítulo 5

Resultados Experimentais

Os resultados de uma bateria de testes realizados sobre os algoritmos da CPref-SQL são apresentados neste capítulo. Testamos a performance e escalabilidade do operador ***Select-Best*** (SB) comparando consultas-pc sob as abordagens de Ordem de preferência ForTe (OFT) e Ordem de preferência FRaca (OFR). Além do mais, comparamos a performance das consultas-pc traduzidas em SQL com recursão. Neste caso, as consultas SQL correspondem semanticamente às consultas CPref-SQL sob a ordem forte, e portanto, retornam sempre o mesmo resultado das consultas com essa abordagem. Avaliamos também o operador ***SelectK-Best*** (SKB) sob as ordens fraca e forte, variando o parâmetro K na cláusula de preferência das consultas.

Primeiramente, na Seção 5.1, detalhes do ambiente experimental, tais como o *benchmark* e critérios para criação das preferências, são descritos. Nas Seções 5.2 e 5.3 apresentamos as análises de performance e escalabilidade dos algoritmos, respectivamente. Na Seção 5.4 ilustramos, ainda, os resultados obtidos com a implementação *on-top* dos algoritmos da linguagem CPref-SQL. Ao final, na Seção 5.5, descrevemos as considerações finais em relação aos resultados obtidos.

5.1 Ambiente Experimental

Todos os experimentos foram executados em um computador AMD Turion X2 Ultra Dual-Core, com frequência de processamento de 2.2GHz, 4GB de memória RAM e HD de 7200rpm, 3Gbps para I/O e 8MB de *buffer*. Utilizamos o sistema operacional Linux Ubuntu 9.10. Os códigos foram desenvolvidos em linguagem C, sobre o SGBDR PostgreSQL 8.4. A aplicação *psql* foi utilizada como *front-end* para o SGBDR, conectada localmente ao servidor durante a submissão das consultas.

A seguir descrevemos a base de dados utilizada para realizar experimentos, bem como as características das preferências e consultas que criamos e executamos.

5.1.1 Base de Dados

Os testes foram realizados utilizando o *benchmark* TPC-H [TPC 2010]. TPC-H é um benchmark específico para consultas *ad-hoc* e suporte a tomadas de decisão. Seu esquema consiste de oito tabelas que modelam um ambiente típico de vendas, conforme ilustra a Figura 5.1. Tabelas como *customer*, *part* e *partsupp* contêm informação sobre itens que empresas de vendas a varejo compram de seus fornecedores e vendem para seus clientes, enquanto *nation* e *region* são pequenas tabelas contendo poucas tuplas. Essas tabelas correspondem a aproximadamente 15% do banco de dados. As maiores tabelas, *lineitem* e *orders* contribuem para os 85% restantes do tamanho total da base de dados.

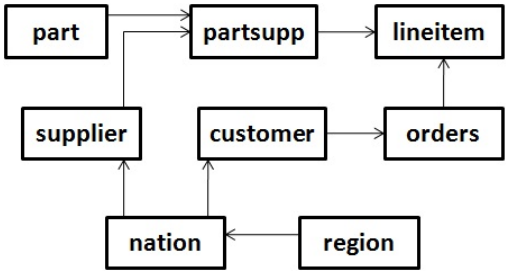


Figura 5.1: Esquema do TPC-H

O *benchmark* TPC-H fornece um gerador de base de dados (dbgen) que cria tuplas nas tabelas do TPC, baseado num fator de escala FE. Tal fator especifica o tamanho do banco de dados. Por exemplo, se FE = 1, então o tamanho total da base será de 1GB. O tamanho das tabelas, exceto *nation* e *region*, é proporcional ao fator de escala. Dessa forma, as bases usadas em nossos experimentos foram geradas com FE = 0.001, 0.005, 0.01, 0.05 e 0.1. Logo, os tamanhos dos nossos bancos são de 1MB, 5MB, 10MB, 50MB e 100MB, respectivamente. O banco *default* tem 10MB.

5.1.2 Consultas

Além da base de dados, o TPC-H fornece um conjunto consultas sobre tal base. As consultas do *benchmark* foram adaptadas para o contexto de preferências a fim de serem usadas nos nossos experimentos. Tal adaptação foi feita utilizando os seguintes critérios:

1. remoção das funções de agregação, uma vez que não são suportadas pela atual implementação da CPref-SQL;
2. inserção da cláusula de preferência (`ACCORDING TO PREFERENCES`) e
3. mudanças nos termos da cláusula `WHERE` para variar o fator de redução do operador de seleção.

Dentre um total de 22 consultas do *benchmark*, selecionamos as consultas Q3, Q5, Q10 e Q18 para utilizar nos experimentos. A escolha de apenas quatro consultas deve-se

ao fato de muitas delas serem parecidas em tratando-se do fator de redução do operador de seleção, o que não acrescentaria mais informações aos nossos testes. As escolhidas possibilitaram uma variação expressiva no número de tuplas submetidas ao operador de preferências. Mais precisamente: 215, 2333, 5756 e 10606 tuplas foram submetidas ao ***Select-Best*** e ***SelectK-Best***, através das consultas Q3, Q5, Q10 e Q18, respectivamente. A consulta Q5 foi considerada como *default*.

5.1.3 Preferências

As teorias de preferência condicional foram construídas levando em consideração duas características: o número de regras e o nível de profundidade.

O número de regras varia de 2 a 40 regras. Criamos preferências com 2, 6, 10, 20, 30 e 40 regras-pc.

O nível de profundidade diz respeito à composição de regras de preferência através do fecho transitivo. Por exemplo, seja $\Gamma = \{\varphi_1, \varphi_2\}$, onde $\varphi_1 : A = a_1 \wedge C = c_1 \rightarrow B = b_1 > B = b_2[E]$ e $\varphi_2 : C = c_1 \wedge D = d_1 \rightarrow B = b_2 > B = b_3[E]$. O nível de profundidade de Γ é 2, uma vez que o tamanho máximo de um caminho induzido pelas regras-pc é 2. Um caminho induzido por regras-pc é uma ordem de preferência sobre tuplas, induzida pela composição de duas ou mais regras-pc. Nesse exemplo, o caminho tem o formato $(a_1, b_1, c_1, d_1, X) > (a_1, b_2, c_1, d_1, Y) > (a_1, b_3, c_1, d_1, Z)$. O nível de profundidade das teorias-pc que utilizamos nos experimentos varia de 1 a 6.

Os antecedentes das regras-pc consideradas nos testes são sempre uma composição booleana de duas fórmulas atômicas, ou seja, as regras são da forma $(P_1(A_1) \wedge P_2(A_2) \rightarrow$ ...).

A teoria-pc *default* (prefDEF) tem 10 regras e nível de profundidade 2. Todas as teorias-pc foram construídas sobre os atributos das oito tabelas do esquema TPC-H.

5.1.4 Tamanho do *Buffer*

Para execução das consultas também levamos em conta a quantidade de memória disponível no *buffer* do SGBDR, já que nossos algoritmos são diretamente dependentes do tamanho da página W de memória (vide Figuras 4.6 e 4.9). Variamos o *buffer* nos tamanhos de 1MB, 2MB, 8MB, 16MB e 28MB (o máximo permitido pelas configurações do sistema operacional). O tamanho *default* é de 8MB.

Por fim, a Tabela 5.1 é uma síntese das características e valores que variamos para realizar nossos experimentos. No total foram seis baterias de teste para compararmos as consultas CPref-SQL de ordem fraca, CPref-SQL de ordem forte e as consultas traduzidas em SQL com recursão.

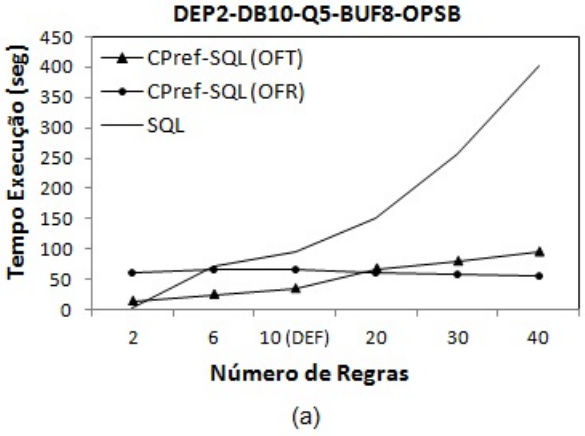
Característica			Default	Variação	Nome da Base
Base de dados	DB	Tamanho da base	10MB	1 a 100MB	PrefDEF-Q5-BUF8-OPSB
Preferências	DEP	Nível de profundidade	2	1 a 6	NRU6-DB10-Q5-BUF8-OPSB
	NRU	Nº de regras	10	2 a 40	DEP2-DB10-Q5-BUF8-OPSB
Consulta	Q	Consulta do TPC-H	Q5	Q3, Q5, Q10, Q18	PrefDEF-DB10-BUF8-OPSB
Buffer	BUF	Memória disponível	8MB	1 a 28MB	PrefDEF-DB10-Q5-OPSB
Operador	OP	Operador de preferência	SB	K: -1 a 5000 (SKB)	PrefDEF-DB10-Q5-BUF8

Tabela 5.1: Bases de dados utilizadas nos experimentos

5.2 Análise de Performance

Na Figura 5.2(a), apresentamos os resultados dos experimentos que comparam as performances da CPref-SQL e SQL quando executamos a consulta Q5, com 8MB de memória disponível. Todas as regras-pc têm nível de profundidade 2 e a quantidade de regras de cada teoria-pc varia de 2 a 40.

Note que a performance da consulta SQL diminui à medida que o número de regras aumenta. Isto pode ser explicado da seguinte maneira: à medida que o número de regras aumenta, mais tuplas podem ser comparadas. Assim, o número de tuplas retornadas pela rotina *MostPref* diminui (ou seja, a possibilidade de duas tuplas serem incomparáveis e ambas preferidas é menor). Se as tuplas preferidas cabem na janela W de *MostPref*, o algoritmo termina em uma ou duas iterações e sua complexidade é $O(n)$, onde n é o tamanho do *input*. Esse não é o caso de uma consulta SQL, que envolve uma junção entre as tabelas `Rules` e `Recursion`.



Número de Tuplas Retornadas

# Regras	OFT/SQL	OFR
2	2291	2243
6	2227	2177
10	1906	1850
20	1859	1804
30	1846	1790
40	1719	1661

(b)

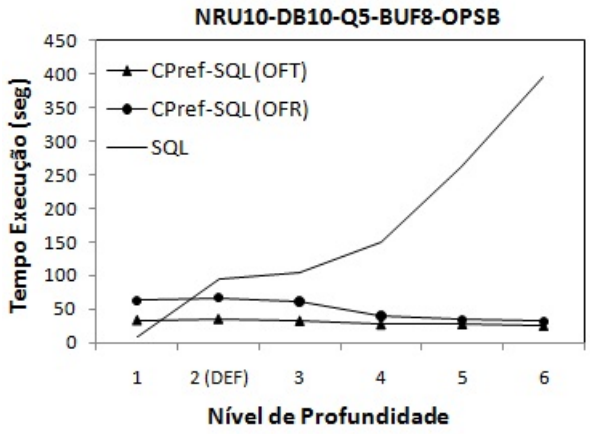
Figura 5.2: Performance e número de tuplas retornadas variando número de regras

Em relação às diferentes ordens das consultas CPref-SQL, de um modo geral, a rotina *weakDomTest* é mais onerosa que a *strongDomTest*, devido aos passos de construção dos conjuntos Δ e $inf(\Delta)$. A diferença de performance entre a ordem forte e a ordem fraca pode ser explicada pela influência do número de regras no número de tuplas retornadas. A Figura 5.2(b) descreve a quantidade de linhas retornadas em cada consulta. A ordem fraca é mais relaxada e pode comparar mais tuplas que a ordem forte. Assim, ela consegue um melhor equilíbrio entre as tuplas preferidas e o número de regras. Para 20, 30 e 40

regras, por exemplo, as consultas OFR tiveram uma performance melhor que as demais.

Na Figura 5.3(a) ilustramos o comportamento das execuções de consultas quando variamos o nível de profundidade das teorias-pc. Esta característica tem um alto impacto na performance da consulta traduzida. À medida que o nível de profundidade aumenta, a tabela `Recursion` tem mais tuplas, e, portanto, a performance da Q5 SQL diminui (devido a junção entre `Rules` e `Recursion`).

Assim como na variação da quantidade de regras, ao aumentar o nível de profundidade, a diferença entre o número de tuplas retornadas pelas consultas OFR e as outras consultas também aumenta, conforme mostra a Figura 5.3(b). A tendência é que a ordem fraca tenha uma performance bem melhor com menos tuplas retornadas. Daí o equilíbrio quando o nível de profundidade é 6.



Número de Tuplas Retornadas

Nível Prof.	OFT/SQL	OFR
1	1930	1874
2	1906	1850
3	1900	1845
4	1842	1457
5	1687	1302
6	1664	1279

(b)

Figura 5.3: Performance e número de tuplas retornadas variando nível de profundidade das teorias-pc

As características de preferências consideradas nas Figuras 5.2 e 5.3 mostram que as performances da SQL e CPref-SQL dependem do fator de seletividade das regras, ou seja, o número de tuplas retornadas pela rotina *MostPref*. Enquanto o número de regras e o nível de profundidade de uma teoria-pc têm um impacto consideravelmente negativo na performance SQL, por outro lado, o desempenho das consultas CPref-SQL não foi fortemente afetado pela variação desses dois parâmetros.

Na Figura 5.4(a) podemos ver como a quantidade de memória principal disponível afeta a performance das execuções de consultas com preferências. Variamos o tamanho do *buffer* de memória de 1MB (1% do tamanho da base) a 28MB (valor *default* do SGBDR e maior que 100% do tamanho da base *default* que utilizamos (10MB)). Claramente, a performance da consulta CPref-SQL apresentou uma oscilação em relação à quantidade de *buffer* disponível. Isto se deve ao fato de que o G-BNL é diretamente afetado pelo tamanho da janela W. Note que a partir de 8MB, o tempo de execução da CPref-SQL é praticamente constante, uma vez que todas as tuplas mais preferidas cabem na memória

principal (dentro da janela W).

Também testamos a performance das consultas-pc top-k variando o parâmetro K (a quantidade desejada de tuplas preferidas), conforme apresentado na Figura 5.4(b). A performance do GRank-BNL é praticamente indiferente para variação dos valores de $K > 0$, uma vez que, em média, o algoritmo varre apenas a metade da relação para obter as K tuplas mais preferidas. Como esperado, a performance para $K = -1$ é melhor, já que para $K > 0$ o algoritmo precisa de um tempo maior para processar os diferentes níveis.

Entre as duas abordagens da CPref-SQL, nossa consulta *default* (DEF) teve uma performance melhor para a ordem de preferência forte. Conforme mencionamos anteriormente, *strongDomTest* é um algoritmo mais rápido que o *weakDomTest*. Assim, apesar da diferença entre o número de tuplas retornadas: 1906 para DEF OFT e 1850 para DEF OFR, o primeiro chega a ser até 2 vezes mais rápido que o segundo.

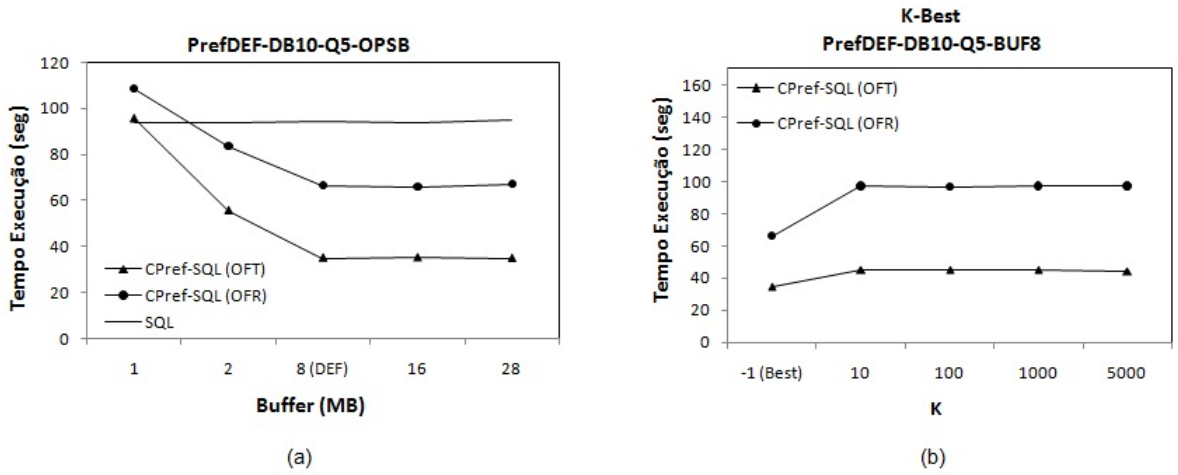
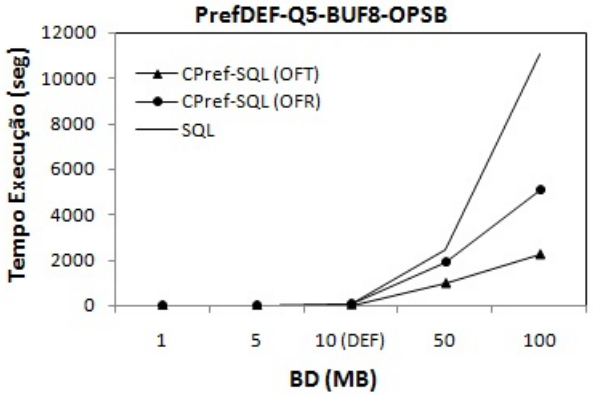


Figura 5.4: (a) Performance variando o buffer. (b) Performance do ***SelectK-Best***

5.3 Análise de Escalabilidade

A Figura 5.5(a) mostra como as consultas SQL e CPref-SQL escalam quando o tamanho da base de dados aumenta desde 1MB até 100MB. Novamente, notamos que a CPref-SQL é mais eficiente que a SQL. A Figura 5.5(b) ilustra o número de tuplas mais preferidas de cada consulta. Claramente, a quantidade de tuplas dominantes afeta o comportamento das consultas. À medida que essa quantidade aumenta, a performance da consulta SQL diminui numa taxa muito maior que as consultas em CPref-SQL.

Na Figura 5.6(a) podemos ver o comportamento das duas linguagens de consulta quando variamos o fator de redução da cláusula `WHERE`, que é o número de tuplas diretamente submetidas ao operador de preferência (após as operações de junção e seleção, de acordo com o plano de execução canônico da Figura 3.11, Capítulo 3). Como menci-



(a)

Número de Tuplas Retornadas

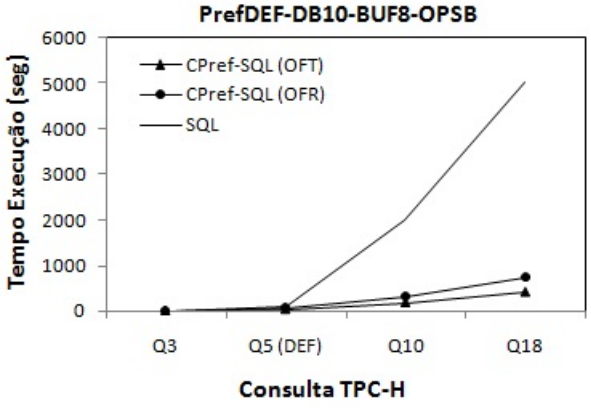
BD (MB)	OFT/SQL	OFR
1	240	202
5	229	187
10	1906	1850
50	9883	9872
100	51329	51327

(b)

Figura 5.5: Escalabilidade e número de tuplas retornadas variando tamanho da base de dados

onado anteriormente, as consultas Q3, Q5, Q10 e Q18 retornam 215, 2333, 5756 e 10606 tuplas, respectivamente, quando executadas sobre os valores *default* sem a cláusula de preferência. A performance da CPref-SQL é bem melhor que as consultas SQL. Note que à medida que o número de tuplas submetidas ao operador de preferência aumenta, ou seja, mais tuplas submetidas à recursão, a performance da SQL decresce drasticamente. Este não é o caso da CPref-SQL: sua performance decrementa numa taxa bem pequena.

Novamente, de acordo com a Figura 5.6(b), consultas CPref-SQL OFR retornam um número menor de tuplas mais preferidas. Entretanto, como o teste de dominância da ordem fraca tem um custo maior para ser computado, a diferença no número de tuplas dominantes não é suficiente. Logo, *strongDomTest* escala melhor.



(a)

Número de Tuplas Retornadas

Consulta	OFT/SQL	OFR
Q3	190	176
Q5	1906	1850
Q10	4765	4708
Q18	7876	7855

(b)

Figura 5.6: Escalabilidade e número de tuplas retornadas variando consulta TPC-H

5.4 Implementação *On-Top*

Executamos diversos experimentos que comparam a implementação *on-top* da CPref-SQL através de funções, com a implementação *built-in* que propusemos nesta dissertação. Além disso, comparamos também o desempenho das consultas em CPref-SQL com o desempenho das consultas traduzidas em SQL com recursão.

Vale lembrar que a CPref-SQL aqui, resolve apenas consultas sob um modelo de preferência *não generalizado*, cujo teste de dominância obedece à semântica de *ordem forte* de preferência. Para efeito de organização, conforme publicamos em [Pereira and de Amo 2010a], os algoritmos utilizados para avaliar os operadores de preferência ***Select-Best*** e ***SelectK-Best*** sob essas condições são denominados BNL** e R-BNL**, respectivamente. A única diferença em relação ao G-BNL e GRank-BNL descritos neste trabalho, é que os primeiros não suportam o modelo generalizado. Entretanto, as complexidades continuam as mesmas. Assim, nestes experimentos, comparamos as implementações dos algoritmos BNL** e R-BNL** sob as abordagens *built-in* e *on-top*.

5.4.1 Ambiente Experimental

O ambiente experimental é praticamente o mesmo daquele que utilizamos nos testes anteriores. Os fatores de variação são: tamanho da base de dados, consultas submetidas, características das preferências e tamanho do *buffer*. Dentre esses, apenas as preferências diferem-se dos experimentos previamente apresentados, já que agora tratamos de uma implementação que suporta apenas preferências com predicados de igualdade (=). A Tabela 5.2 sintetiza as características das bases de dados utilizadas nestes testes.

Característica			**Default**	**Variação**	**Nome da Base**
Base de dados	DB	Tamanho da base	10MB	1 a 100MB	PrefDEF'-Q5-BUF8-OPSB
Preferências	DEP	Nível de profundidade	2	1 a 6	NRU6-DB10-Q5-BUF8-OPSB
	NRU	N° de regras	6	2 a 40	DEP2'-DB10-Q5-BUF8-OPSB
Consulta	Q	Consulta do TPC-H	Q5	Q3, Q5, Q10, Q18	PrefDEF'-DB10-BUF8-OPSB
Buffer	BUF	Memória disponível	8MB	1 a 28MB	PrefDEF'-DB10-Q5-OPSB
Operador	OP	Operador de preferência	SB	K: -1 a 5000 (SKB)	PrefDEF'-DB10-Q5-BUF8

Tabela 5.2: Bases de dados utilizadas nos experimentos para versão *on-top* da linguagem

5.4.2 Análise de Performance

Nas Figuras 5.7(a) e 5.7(b) apresentamos os resultados que comparam a consulta Q5 sendo executada com diferentes preferências. Em ambos os casos, percebemos um comportamento semelhante: as consultas em CPref-SQL são sempre mais ágeis do que as traduções em SQL com recursão. Para 40 regras, por exemplo, o desempenho da implementação *built-in* chega a ser 4 vezes melhor do que a consulta SQL.

Conforme esperado, as consultas *on-top* tiveram uma performance inferior às consultas-pc executadas no *backend* do SGBDR. Apesar do algoritmo BNL** ser o mesmo, as fun-

ções *on-top* têm um *overhead*, por exemplo de uma operação de projeção extra, por serem independentes do código-fonte do gerenciador.

Com a variação do *buffer* disponível, o desempenho das consultas *built-in* continua superior, conforme ilustra a Figura 5.7(c).

Por fim, comparando os algoritmos BNL** e R-BNL** sob as duas formas de implementação, através da Figura 5.7(d), concluímos que a performance das consultas CPref-SQL executadas no *core* do SGBDR chega a ser até 2 vezes melhor.

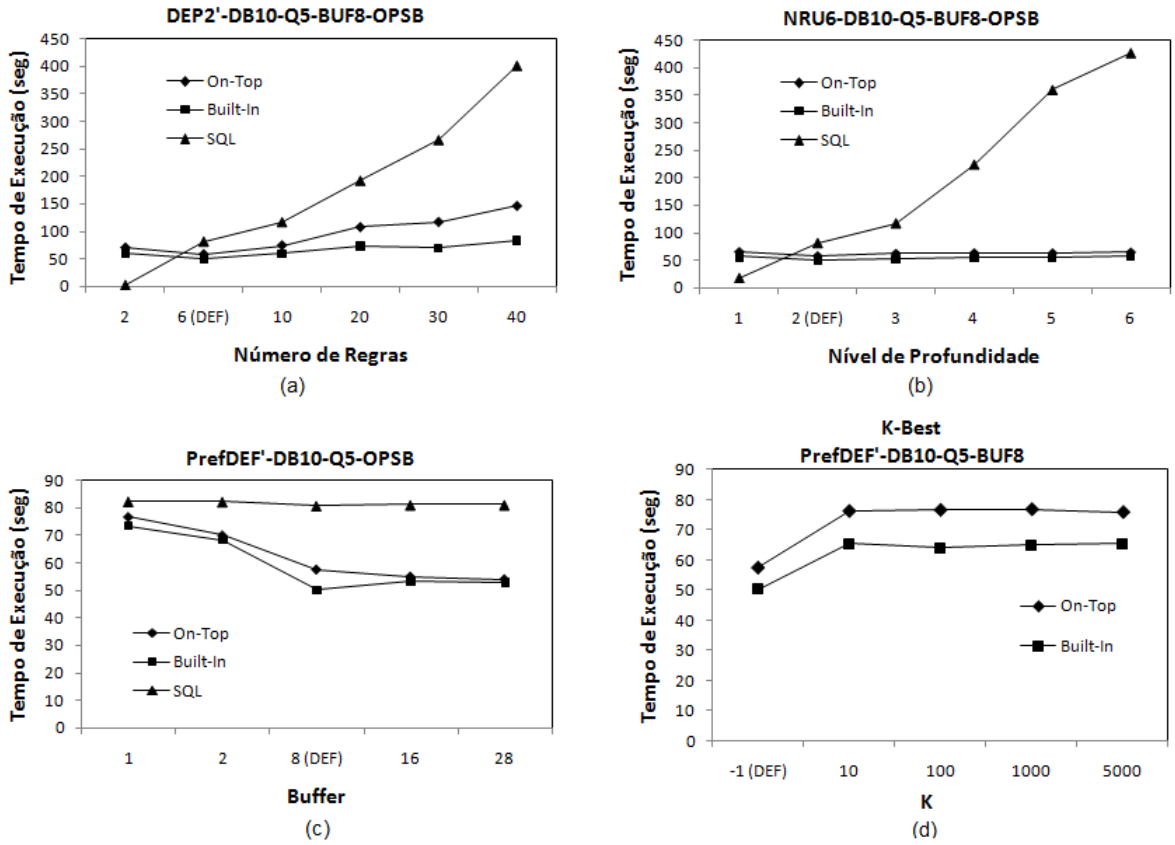


Figura 5.7: Resultados de performance

5.4.3 Análise de Escalabilidade

A Figura 5.8(a) mostra como as consultas escalam quando o tamanho da base varia de 1MB a 100MB. Novamente, notamos que a CPref-SQL, independente da implementação, é mais eficiente que a SQL. A CPref-SQL *built-in*, por sua vez, tem um desempenho bem melhor que a *on-top*. Claramente, o número de tuplas preferidas afeta o comportamento das consultas. À medida que essa quantidade aumenta, a performance das consultas *on-top* e SQL decrescem numa taxa maior do que as consultas *built-in*.

Da mesma maneira acontece quando variamos o número de tuplas submetidas ao operador de preferência. As consultas Q3, Q5, Q10 e Q18 retornam 215, 2333, 5756 e 10606 tuplas, respectivamente, quando executadas sobre a base *default* de 10MB sem a

cláusula de preferência. Conforme ilustra o gráfico da Figura 5.8(b), variando esse fator de redução, a performance SQL é inferior à CPref-SQL. As consultas *on-top*, ainda que bem melhores que as SQL, não superam a implementação abordada neste trabalho: *built-in*.

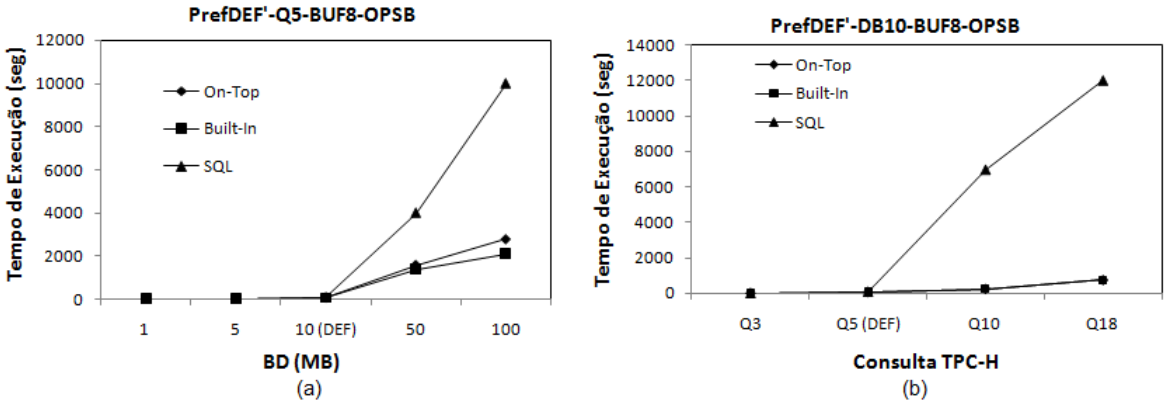


Figura 5.8: Resultados de escalabilidade

5.5 Considerações Finais

Os resultados obtidos foram bastante satisfatórios. Eles mostraram que as consultas CPref-SQL são mais eficientes que a simples expressão de preferências através de SQL recursivo. Concluímos também que, se por um lado a CPref-SQL sob a ordem forte de preferência é mais conservativa, por outro, ela tem um melhor desempenho que a CPref-SQL de ordem fraca, de acordo com os algoritmos que implementamos.

Quanto aos resultados da abordagem alternativa para implementação da CPref-SQL, *on-top*, independente do código-fonte do SGBDR, apesar de ser mais flexível quanto à independência de plataforma e de mais rápido desenvolvimento, possui menor desempenho.

Cabe, portanto, ao usuário definir sua ***preferência*** sobre a melhor implementação da linguagem CPref-SQL, diante de suas necessidades!

Capítulo 6

Estudos de Caso

Uma vez implementada, neste capítulo apresentamos duas aplicações da linguagem CPref-SQL. Tais aplicações estão ligadas à integração entre as abordagens de consultas com preferências e consultas por similaridade. Os estudos de caso apresentados aqui são resultantes de uma participação no Projeto de Cooperação Acadêmica (PROCAD) entre a Universidade Federal de Uberlândia (UFU) e o Instituto de Ciências Matemáticas e de Computação (ICMC - USP São Carlos). Trata-se de um projeto com o objetivo de desenvolver estudos que relacionam operadores de busca e similaridade em ferramentas de apoio ao desenvolvimento de software, especialmente para a área da saúde.

O primeiro estudo de caso, apresentado na Seção 6.1, é uma proposta de combinação entre os operadores de preferência da CPref-SQL e operadores de similaridade propostos em [Barioni et al. 2006], aplicados ao contexto de imagens médicas. Já a Seção 6.2 descreve o segundo estudo de caso, que é uma continuidade da proposta anterior: exploramos uma maneira de obter preferências automaticamente, aproveitando o módulo de mineração de dados existente no sistema de busca por similaridade, discutido em [Ferreira et al. 2010b]. Por fim, na Seção 6.3, expomos as conclusões tiradas acerca das aplicações da linguagem CPref-SQL apresentadas neste capítulo.

6.1 Similaridade e Preferências sobre Imagens Médicas

O desenvolvimento de técnicas de busca por similaridade em imagens tem sido um importante tópico de pesquisa. Imagens de exames médicos, por exemplo, estão sendo cada vez mais armazenadas em sistemas de bancos de dados. Entretanto, por ser avaliada utilizando métodos numéricos, a similaridade pode eventualmente ordenar as imagens de uma maneira diferente daquela desejada pelo usuário.

Diversos grupos de pesquisa na área de recuperação de imagens baseada em conteúdo (*Content-Based Image Retrieval* - CBIR) estudam maneiras de incorporar a percepção de similaridade do médico no momento das consultas [Ponciano-Silva et al. 2009, Rorissa et al. 2008, Kim et al. 2010, Zhou et al. 2008]. De fato, durante a busca por imagens

em um sistema, o conhecimento do radiologista e como ele indica suas preferências é fundamental. Por exemplo, suponha que o especialista esteja procurando por doenças pulmonares em um raio-X, e busca por imagens de casos similares, conforme a consulta Q1 abaixo:

> **Q1:** *''Dentre as 15 imagens mais similares a este raio-X de pulmão, que contenham a palavra Consolidação no relatório, o radiologista prefere aquelas imagens obtidas durante as estações mais secas do ano."*

O SIREN (*SImilarity Retrieval ENgine*), proposto em [Barioni et al. 2006], é um exemplo de sistema CBIR. É um serviço implementado entre o sistema gerenciador de banco de dados relacionais (SGBDR) e a aplicação, que permite a execução de consultas por similaridade em SQL. Tal serviço intercepta e analisa todo comando SQL enviado de uma aplicação, tratando os predicados referentes a similaridade e a dados complexos. Quando um comando não tem operação de similaridade nem dados complexos, o SIREN é transparente e o comando é diretamente enviado ao SGBDR. Se existem predicados de similaridade, o SIREN executa operações e reescreve o comando, enviando para o SGBDR executar as operações tradicionais. Entretanto, originalmente, o SIREN não consegue recuperar imagens que atendam às preferências do usuário.

Propusemos, então, uma extensão do SIREN, adicionando um módulo de preferência à sua arquitetura, de modo que os operadores de similaridade e preferências possam ser combinados, dando origem às consultas por similaridade com preferências. As propostas e resultados discutidas neste estudo de caso foram publicadas no trabalho [Ferreira et al. 2010a].

6.1.1 Módulo de Preferência no SIREN

A Figura 6.1 ilustra a arquitetura do SIREN. A manipulação das consultas por similaridade acontece no otimizador. É nele que as consultas são reescritas utilizando a álgebra de similaridade desenvolvida em [Ferreira et al. 2009] e enviadas ao SGBDR. O módulo de preferência foi desenvolvido como um novo componente funcional do otimizador SIREN.

A seguir descrevemos os passos para criação de preferências e execução de consultas híbridas no sistema. A escolha de uma nova dinâmica, com novos comandos para criação e execução de preferências ocorreu por uma questão de compatibilidade com a estrutura SIREN.

Nesta proposta, um Modelo de Preferência (MP) é o responsável por definir as regras-pc no perfil do usuário. Um novo MP é definido pelo comando `CREATE PREFERENCE MODEL`, cuja sintaxe é similar ao comando `CREATE PREFERENCES` da CPref-SQL:

```
<create_preference_model_statement>::=
    CREATE PREFERENCE MODEL <nome_modelo>
```

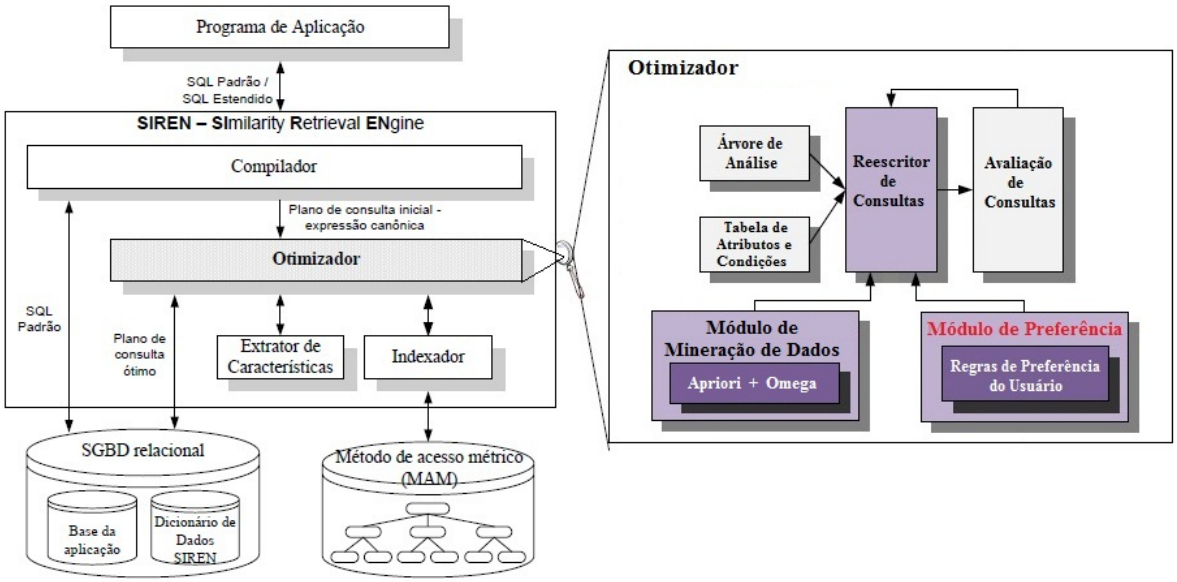


Figura 6.1: Arquitetura SIREN com módulo de preferência

```
FROM <lista_relacoes>
AS <lista_preferencias> [‘[’ <lista_atributos> ‘]’];
```

O novo MP, chamado `<nome_modelo>`, atribui a lista de regras-pc, definida em `<lista_preferencias>`, às relações especificadas em `<lista_relacoes>`. Cada regra é especificada seguindo a mesma sintaxe do comando `CREATE PREFERENCES`, detalhada no Capítulo 3. O parâmetro opcional `<lista_atributos>` exprime os atributos que não se encaixam na semântica *ceteris paribus*, já que, neste estudo de caso, foi utilizada a abordagem de ordem forte de preferência, também definida no Capítulo 3.

É função do módulo de preferência receber tal comando e armazenar as regras no catálogo. O Exemplo 6.1 ilustra a criação de um MP.

Exemplo 6.1. Suponha que um radiologista defina seu perfil levando em consideração doenças pulmonares como bronquiolite e pneumonia. Seu interesse é por tomografias pulmonares que contenham o achado “Consolidação” e que tenham sido tiradas nas épocas mais secas do ano. Logo, o MP será:

```
CREATE PREFERENCE MODEL PulmaoPref
FROM ExamesPulmao AS
    IF class = ‘Consolidacao’ THEN
      season = ‘inverno’ > season = ‘outono’ [id, sexo, idade, data] AND
    IF class = ‘Consolidacao’ THEN
      season = ‘outono’ > season = ‘primavera’ [id, sexo, idade, data] AND
    IF class = ‘Consolidacao’ THEN
      season = ‘primavera’ > season = ‘verao’ [id, sexo, idade, data];
```

Uma vez criado um MP, as regras-pc estão prontas para serem utilizadas. Entretanto, é necessário que o usuário atribua explicitamente o novo modelo ao seu ambiente de

consulta. A sentença `SET MODIFICATION` controla as modificações que podem ser feitas pelo usuário em seu perfil e tem a seguinte sintaxe:

```
<set_modification_statement>::=
    SET MODIFICATION [ADD | REMOVE | UPDATE]
                     [ALL | <model_name>];
```

Este comando é usado para habilitar ou desabilitar um MP (cláusulas `ADD` ou `REMOVE`), ou ainda, para atualizar o MP já existente (`UPDATE`). Tais modificações podem ser feitas para todos os MPs existentes no ambiente do usuário, especificando o parâmetro `ALL` ou apenas para o MP `<model_name>`. Vale ressaltar que a cada execução do comando `SET MODIFICATION` a consistência dos MPs habilitados é validada seguindo os algoritmos implementados pela CPref-SQL.

Exemplo 6.2. Para habilitar o MP `PulmaoPref` definido no Exemplo 6.1 ao ambiente do usuário, basta executar o comando:

```
SET MODIFICATION ADD PulmaoPref;
```

Assim, todas as consultas subsequentes serão reescritas levando em consideração o MP `PulmaoPref`.

A reescrita de uma consulta que tenha o MP habilitado é feita adicionando a cláusula `ACCORDING TO PREFERENCES`, cuja sintaxe é a mesma da CPref-SQL:

```
<according_clause>::= ACCORDING TO PREFERENCES
                      ([n, ] <model_name>);
```

Essa cláusula permite que, após a execução das cláusulas `FROM` e `WHERE`, seja feito um filtro adicional à resposta, levando em consideração as preferências do usuário. O parâmetro opcional n, que é definido no perfil do usuário, permite que sejam selecionadas as n tuplas mais preferidas.

O operador utilizado para as consultas por similaridade neste estudo de caso foi o *k-nearest neighbor* (kNN), que calcula os k vizinhos mais próximos do centro de consulta [B.W. Silverman and Hodges 1989]. A semântica de uma consulta k-NN com preferências corresponde a: primeiro selecione as k imagens mais similares à imagem de referência; depois, dentre as resultantes, selecione até n imagens mais preferidas, caso o parâmetro n tenha sido especificado, ou apenas as mais preferidas caso contrário.

Exemplo 6.3. Para executar a consulta **Q1** no SIREN, o seguinte comando é submetido ao sistema:

```
    SELECT id, idade, Image
```

```
FROM ExamesPulmao
WHERE Image NEAR 'C:\Exame1.jpg' STOP AFTER 15;
```

Supondo que o MP `PulmaoPref` esteja habilitado no perfil do médico, a reescrita é feita e dá origem ao seguinte comando:

```
SELECT id, idade, Image
FROM ExamesPulmao
WHERE Image NEAR 'C:\Exame1.jpg' STOP AFTER 15
ACCORDING TO PREFERENCES (PulmaoPref);
```

Tal consulta retornará as 15 imagens mais similares à imagem de referência e, dentre elas, as mais preferidas de acordo com a hierarquia de preferências do radiologista.

O módulo de preferência do SIREN controla, portanto, a criação, gerência e utilização de um modelo de preferência no SIREN.

6.1.2 Avaliação Experimental

O SIREN e seu módulo de preferência estão implementados em C++ e os experimentos foram avaliados utilizando um processador Intel Core 2 Quad 2.83GHz, com 4GB de memória RAM, sob o sistema operacional Windows XP. O SGBDR utilizado foi o PostgreSQL 8.4 estendido com a implementação da linguagem CPref-SQL sob a ordem forte de preferência.

Os experimentos realizados comparam a execução de consultas por similaridade no SIREN que utilizam ou não regras de preferência.

A base de dados é formada por 246 imagens de pulmão, provenientes de 108 tomografias computadorizadas de pacientes do Hospital das Clínicas de Ribeirão Preto (HCRP) da Universidade de São Paulo (USP). Cada imagem foi classificada por um radiologista em seis classes distintas de acordo com os achados radiológicos: Consolidação, Enfisema, Espessamento, Favo-de-mel, Vidro Fosco e Normal. Em média, são 40 imagens por classe.

Foram usados dois perfis diferentes nos experimentos: um genérico, ou seja, com regras de preferência menos restritivas e outro mais específico, com regras condicionais. O MP mais específico é o `PulmaoPref`, descrito no Exemplo 6.1. O mais genérico, denominado `GenPulmaoPref` é definido no Exemplo 6.4.

Exemplo 6.4.

```
CREATE PREFERENCE MODEL GenPulmaoPref
FROM ExamesPulmao AS
  estacao='inverno' > estacao='outono' [id, classe, sexo, idade, data] AND
  estacao='outono' > estacao='primavera' [id, classe, sexo, idade, data] AND
  estacao='primavera' > estacao='verao' [id, classe, sexo, idade, data];
```

A consulta k-NN **Q1**, codificada como no Exemplo 6.3 foi submetida ao SIREN com $k = 50$ para 82 consultas com imagens de referência distintas, cobrindo, portanto, 1/3 da base total. Cada consulta foi executada 3 vezes: com o MP desabilitado e com os MPs `PulmaoPref` ou `GenPulmaoPref` habilitados. A Figura 6.2 apresenta a porcentagem de respostas interessantes às consultas. Uma resposta interessante é aquela que tem a mesma classe que a imagem de referência. No gráfico, tal porcentagem é a proporção entre as imagens relevantes em relação às imagens de mesma classe do banco de dados.

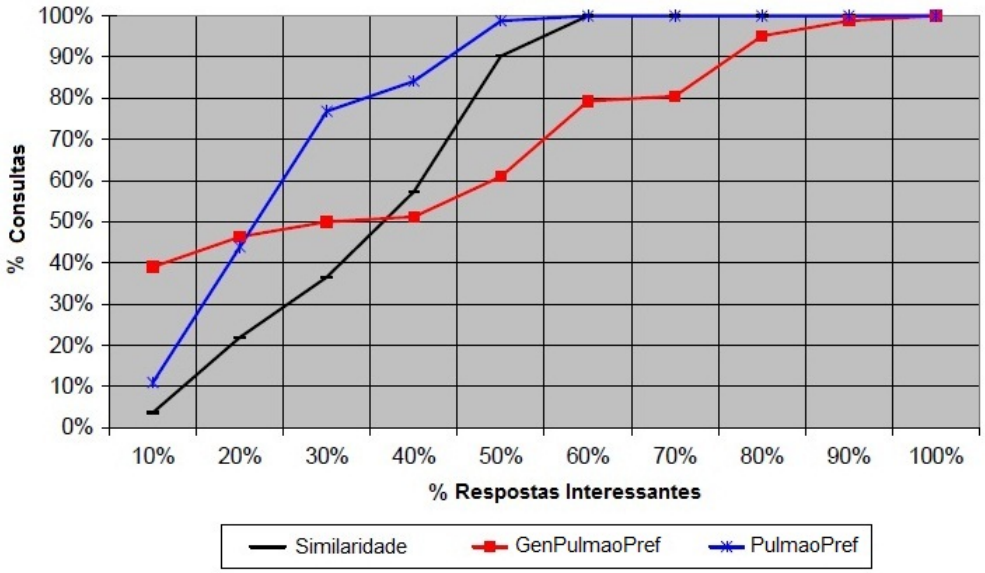


Figura 6.2: Porcentagem de respostas corretas em consultas por similaridade e preferências

Analisando os resultados, é possível perceber que as preferências aumentaram a quantidade de consultas que retornam imagens interessantes. Por exemplo, apenas 37% das consultas por similaridade puras conseguiram recuperar 30% de imagens interessantes, enquanto 50% das consultas com o MP `GenPulmaoPref` e 77% das consultas com o MP `PulmaoPref` conseguiram recuperar os mesmos 30% de respostas interessantes. Por outro lado, o gráfico mostra também que se o MP é muito genérico, torna-se incapaz de melhorar as respostas para grande quantidade de imagens retornadas. De fato, 90% das consultas sem preferências e 100% das consultas com o MP menos genérico retornaram até 50% de respostas corretas, já a consulta com o MP `GenPulmaoPref` obteve apenas 61% de respostas interessantes. Isso acontece porque as preferências genéricas têm chance de retornar muito mais imagens de classes distintas daquela original.

6.1.3 Conclusões

Neste estudo de caso apresentamos uma nova técnica para melhorar a qualidade das consultas por similaridade em bases de dados de imagens, incorporando respostas que atendem às expectativas dos usuários. Os experimentos usando o protótipo do SIREN

mostraram que adicionar preferências às consultas por similaridade possibilitou obter mais respostas interessantes. É, portanto, uma aplicação da linguagem CPref-SQL que, através de consultas híbridas, contribui para melhor qualidade nas respostas.

6.2 Mineração de Preferências em Consultas por Similaridade

O SIREN possui um Módulo de Mineração de Dados (MMD) que minera dados complexos, gerando regras de associação e utilizando-as para melhorar a qualidade e velocidade das consultas por similaridade [Ferreira et al. 2010b].

Neste estudo de caso, propomos utilizar o MMD do SIREN para obter automaticamente as regras de preferência. Mais especificamente, propomos uma técnica de tradução das regras de associação mineradas pelo SIREN em regras-pc. É, portanto, um complemento ao estudo de caso anterior, onde agora o próprio sistema encarrega-se de elicitar e criar um modelo de preferência.

6.2.1 Módulo de Mineração de Dados do SIREN

O MMD foi desenvolvido como um componente funcional do SIREN por [Ferreira et al. 2010b] (Figura 6.1). Ele extrai o conhecimento da base multimídia e gera regras de associação, correlacionando a informação semântica dos dados texto às características extraídas dos dados complexos. Os algoritmos *Apriori* [Agrawal and Srikant 1994] e *Omega* [Ribeiro et al. 2008], referentes à geração de regras de associação e à manipulação de vetores de características, respectivamente, estão implementados no MMD.

O objetivo do MMD é, portanto, tirar proveito da álgebra de similaridade implementada no reescritor de consultas do SIREN [Ferreira et al. 2009], e usar regras de associação para aprimorar a reescrita e melhor atender às expectativas do usuário.

A dinâmica do MMD é a seguinte:

1. Para que uma base de dados de imagens esteja apta a ser usada no MMD, toda imagem deve ter uma *tag* (classe), proveniente de um conjunto de palavras-chave, que a descreva. É o usuário que classifica a imagem quando ela é inserida.
2. A base de dados também deve ter um atributo que indica se a imagem deverá fazer parte ou não de um subconjunto de treino que será utilizado no passo da mineração. Quem indica se a imagem será de treino ou não também é o usuário no momento da inserção.
3. Dada uma base preparada, ou seja, com todas as imagens classificadas e algumas setadas para treinamento, o usuário cria um modelo de mineração, no qual define basicamente a base de dados sobre a qual o modelo atua, o atributo que indica

as imagens que formarão o subconjunto de treino e o algoritmo de mineração que deseja utilizar. Um modelo de mineração é criado através do comando `CREATE MINING MODEL`.

4. O modelo de mineração criado gera regras de associação do tipo:

$$f_{r_1}[l_{1_0} - l_{1_1}], ..., f_{r_n}[l_{n_0} - l_{n_1}] \rightarrow Classe_{R_1}, ..., Classe_{R_m}\ (sup, conf),$$

onde imagens que têm as características $f_{r_1}, ..., f_{r_n}$ respectivamente nos intervalos $[l_{1_0} - l_{1_1}], ..., [l_{n_0} - l_{n_1}]$ tendem a ser das classes $Classe_{R_1}, ..., Classe_{R_m}$. O número de características no antecedente da regra é $1 \leq n \leq max_{carac}$, onde max_{carac} é a maior quantidade de características que pode ser extraída pelo extrator. O número de classes no consequente é $1 \leq m \leq max_{classe}$, onde max_{classe} é o número de classes encontradas na relação. Os valores sup e $conf$ indicam, respectivamente, o suporte e a confiança para a regra.

5. Caso o usuário queira utilizar o modelo de preferência, deve habilitá-lo com o comando `SET MODIFICATION ADD <nome_modelo>`.

6. Toda consulta por similaridade agora é reescrita adicionando à cláusula `WHERE` condições do tipo `AND (Tag=<cl`$_1$`> OR ... OR Tag=<cl`$_m$`>)`, onde cada $<\text{cl}_i>$ para $i \in \{1, ..., m\}$ é uma classe que aparece no consequente de uma regra, cujos antecedentes satisfazem as características da imagem de consulta.

O exemplo a seguir ilustra os passos de criação e funcionamento de um modelo de mineração:

Exemplo 6.5. Suponha uma relação `MaravilhasDoMundo` que armazena fotos das novas e antigas maravilhas do mundo. A Figura 6.3 ilustra uma instância dessa relação.

TAG	TREINO	PAÍS	CIDADE	IMAGE
ChichenItza	True	México	Chichen Itza	C:\img1.jpg
CristoRedentor	False	Brasil	Rio de Janeiro	C:\img4.jpg
Coliseu	False	Italia	Roma	C:\img9.jpg
CristoRedentor	False	Brasil	Rio de Janeiro	C:\img11.jpg
TajMahal	True	India	Mara	C:\img2.jpg

Figura 6.3: Instância da relação `MaravilhasDoMundo`

O atributo `Tag` refere-se à classe que a imagem pertence. O domínio desse atributo contém as 15 maravilhas do mundo, entre as novas e antigas mais o complexo de Gizé. O atributo `Treino` indica se a imagem faz parte do subconjunto de treino que será usado no minerador. O atributo complexo `Image` armazena a imagem e o restante dos atributos texto descrevem seus detalhes.

Como a base `MaravilhasDoMundo` está preparada com tuplas de treino e imagens classificadas, suponha que um modelo de mineração seja criado com o seguinte comando:

```
CREATE MINING MODEL ModeloMaravilhasDoMundo
```

```
ON MaravilhasDoMundo (Image) METRIC textura
WHERE Treino='True'
USING APRIORI (Tag, 15, 1, 100, 2, 0.1, 0.42);
```

Nesse comando, o modelo utiliza o algoritmo *Apriori* para extrair regras de associação do atributo `Image`. O *Apriori* processará apenas as tuplas cujo atributo `Treino` é `'True'`. Os parâmetros do algoritmo representam, respectivamente: o atributo usado na classificação das imagens (`Tag`), número de classes existentes (15), suporte e confiança das regras (1% e 100%). O três últimos parâmetros são opcionais para calibrar o algoritmo *Omega* [Ribeiro et al. 2008].

Uma regra de associação R gerada pelo modelo `ModeloMaravilhasDoMundo` é do tipo:

$$70[836.1 - 877.6]\ 8[155.7 - 156.5] \rightarrow ChichenItza\ (1.0, 100.0)$$

Ou seja, as imagens que tenham suas 70ª e 8ª características com valores nos intervalos fechados [836.1-877.6] e [155.7-156.5], respectivamente, tendem a ser imagens de *ChichenItza*.

Assim, suponha que a consulta abaixo seja submetida ao SIREN:

```
SELECT Pais, Image
FROM MaravilhasDoMundo
WHERE Image NEAR 'C:\Img1.jpg' STOP AFTER 15;
```

Com o `ModeloMaravilhasDoMundo` habilitado, o minerador de dados entrará em ação e, se as 70ª e 8ª características da imagem de referência satisfizerem os intervalos da regra R, por exemplo, a consulta reescrita resultante será do tipo:

```
SELECT Pais, Image
FROM MaravilhasDoMundo
WHERE Image NEAR 'C:\Img1.jpg' STOP AFTER 15
      AND Tag='ChichenItza';
```

6.2.2 Extração de um Modelo de Preferência

Nossa proposta neste estudo de caso é transformar as regras de associação mineradas pelo MMD do SIREN em regras-pc que darão origem a um novo Modelo de Preferência (MP). Assim, toda vez que um modelo de mineração é criado, ocorre a tradução das regras e, automaticamente, um novo MP também é criado.

A tradução de uma regra de associação em regras de preferência condicional da CPref-SQL é feita da seguinte maneira:

- Por questões de semântica das regras-pc, os valores das características dos atributos devem ser categorizados. A categoria CX de um valor V é dada por: $X = \lfloor V/P \rfloor$,

onde P é um parâmetro pré-estabelecido. Neste estudo de caso, fixamos $P = 100$. Então, a categoria $C0$ corresponde aos valores pertencentes ao intervalo fechado $[0 - 99]$, $C2 = [100 - 199]$ e assim por diante. Tomando como exemplo a regra $70[836.1 - 877.6]\ 8[155.7 - 156.5] \rightarrow ChichenItza\ (1.0, 100.0)$, temos que os valores da característica 70 correspondem à categoria $C8$ e a característica 8 deve ter seus valores na categoria $C1$.

Caso um intervalo contenha valores que pertençam a categorias diferentes, apenas a categoria do limite inferior é considerada. Por exemplo, suponha que uma regra r tenha o predicado antecedente $3[184.3 - 215.8]$, ou seja, r é satisfeita se a característica 3 tiver seu valor pertencente ao intervalo $[184.3 - 215.8]$. Como os valores 184.3 e 215.8 pertencem às categorias C1 e C2, respectivamente, durante a tradução, apenas a categoria C1 será levada em conta.

Vale ressaltar que a escolha do parâmetro $P = 100$ neste estudo de caso foi aleatória. Cálculos estatísticos podem ser feitos para analisar os valores envolvidos nas regras de associação e obter a melhor divisão por categorias, seguindo critérios como homogeneidade, precisão, entre outros.

- Uma vez estabelecidas as categorias, seja $R : f_{r_1}[l_{1_0} - l_{1_1}], ..., f_{r_n}[l_{n_0} - l_{n_1}] \rightarrow Classe_{R_1}, ..., Classe_{R_m}\ (sup, conf)$ uma regra de associação minerada pelo SIREN. R dá origem às seguintes regras-pc:

$$f_{r_1} = C_{I_1} \wedge ... \wedge f_{r_n} = C_{I_n} \rightarrow Tag = Classe_{R_1} > Tag = Classe_{R_{j_0}}\ [\mathrm{W}]$$
$$f_{r_1} = C_{I_1} \wedge ... \wedge f_{r_n} = C_{I_n} \rightarrow Tag = Classe_{R_1} > Tag = Classe_{R_{j_1}}\ [\mathrm{W}]$$
$$...$$
$$f_{r_1} = C_{I_1} \wedge ... \wedge f_{r_n} = C_{I_n} \rightarrow Tag = Classe_{R_1} > Tag = Classe_{R_{j_l}}\ [\mathrm{W}]$$
$$f_{r_1} = C_{I_1} \wedge ... \wedge f_{r_n} = C_{I_n} \rightarrow Tag = Classe_{R_2} > Tag = Classe_{R_{j_0}}\ [\mathrm{W}]$$
$$f_{r_1} = C_{I_1} \wedge ... \wedge f_{r_n} = C_{I_n} \rightarrow Tag = Classe_{R_2} > Tag = Classe_{R_{j_1}}\ [\mathrm{W}]$$
$$...$$
$$f_{r_1} = C_{I_1} \wedge ... \wedge f_{r_n} = C_{I_n} \rightarrow Tag = Classe_{R_2} > Tag = Classe_{R_{j_l}}\ [\mathrm{W}]$$
$$.$$
$$.$$
$$.$$
$$f_{r_1} = C_{I_1} \wedge ... \wedge f_{r_n} = C_{I_n} \rightarrow Tag = Classe_{R_m} > Tag = Classe_{R_{j_l}}\ [\mathrm{W}]$$

onde: $\{Classe_{R_{j_0}}, ..., Classe_{R_{j_l}}\} \cap \{Classe_{R_1}, ..., Classe_{R_m}\} = \emptyset$ e $\{Classe_{R_{j_0}}, ..., Classe_{R_{j_l}}\} \cup \{Classe_{R_1}, ..., Classe_{R_m}\}$ = todas as *tags* (classes) do domínio. C_{I_k} corresponde à categoria do *limite inferior* da característica f_{r_k}, para $k \in \{1, ..., n\}$. O conjunto W corresponde a todos os atributos da relação S em questão que não fazem parte da regra R e que não geram inconsistências, conforme detalharemos no próximo tópico. De um modo geral,

$W = Atr(S) - \{f_{r_1}, ..., f_{r_n}, Tag\}$.

- Para garantir que não haja inconsistências na teoria-pc formada, os seguintes critérios foram adotados:

  1. Consistência local. Ao gerar uma nova regra r, verificamos se ela não criará inconsistência local sobre o atributo `Tag` em relação às regras obtidas anteriormente. Se for detectada uma inconsistência, r é descartada.
  2. Grafo de dependência. Para garantir que o grafo de dependência dos atributos seja acíclico e, ao mesmo tempo, relaxarmos ao máximo a semântica das regras, adotamos o seguinte critério durante a criação do conjunto W de uma regra r: para todo atributo A tal que $A \in Atr(S) - \{f_{r_1}, ..., f_{r_n}, Tag\}$, A é inserido em W. Se A gerar ciclo no grafo de dependência da teoria-pc obtida até então, A é removido de W.

  Note que devido tais critérios de consistência, a tradução das regras torna-se dependente da ordem em que são geradas. Ou seja, teorias-pc diferentes podem ser obtidas apenas com um rearranjo na ordem de organização das regras de associação.

O exemplo a seguir ilustra todo o processo de tradução de regras para criação de um novo Modelo de Preferência no SIREN.

Exemplo 6.6. Suponha que durante a criação do modelo de mineração de dados `ModeloMaravilhasDoMundo` sobre a relação `MaravilhasDoMundo` do Exemplo 6.5, as seguintes regras de associação tenham sido geradas:

1. $5[356.9 - 398.1]\ 1[22.5 - 111.7] \rightarrow ChichenItza\ (1.0, 100.0)$
2. $2[427.5 - 510.1] \rightarrow TajMahal\ (1.0, 100.0)$

Para simplificar, considere que o domínio do atributo `Tag` seja formado apenas pelo conjunto {ChichenItza, CristoRedentor, Coliseu, TajMahal} e que o vetor de características das imagens seja composto por 5 posições. De acordo com nossa proposta para tradução das regras de associação, automaticamente após a criação do modelo de mineração de dados, o comando `CREATE PREFERENCE MODEL` será submetido ao sistema. Teremos, então:

```
CREATE PREFERENCE MODEL PrefModeloMaravilhasDoMundo
FROM ProjMaravilhasDoMundo AS
 IF 5=C3 and 1=C0 THEN Tag='ChichenItza' > Tag='CristoRedentor' [2,3,4] and
 IF 5=C3 and 1=C0 THEN Tag='ChichenItza' > Tag='Coliseu' [2,3,4] and
 IF 5=C3 and 1=C0 THEN Tag='ChichenItza' > Tag='TajMahal' [2,3,4] and
 IF 2=C4 THEN Tag='TajMahal' > Tag='CristoRedentor' [3,4] and
```

```
IF 2=C4 THEN Tag='TajMahal' > Tag='Coliseu' [3,4];
```

A relação em questão, `ProjMaravilhasDoMundo`, é uma projeção da relação original `MaravilhasDoMundo` sobre o vetor de características e o atributo `Tag`. Ou seja: `ProjMaravilhasDoMundo(Tag,1,2,3,4,5)`.

Vale ressaltar que os módulos de mineração de dados e de preferência são mutuamente exclusivos. Portanto, quando o usuário, através do comando `SET MODIFICATION`, opta por um modelo de preferência que tem um modelo de mineração correspondente, o de mineração é automaticamente desabilitado e vice-versa.

6.2.3 Resultados Experimentais

O sistema de tradução de regras foi implementado no Módulo de Preferência do SIREN em linguagem C++. O SGBDR utilizado é o PostgreSQL 8.4 estendido com a CPref-SQL implementada sob a ordem de preferência forte. O objetivo dos testes é comparar a acurácia de consultas por similaridade sob três abordagens: (1) sem ativação dos módulos de mineração de dados (MMD) e preferência (MP), (2) com ativação do MMD e (3) com ativação do MP.

Base de Dados. Os experimentos foram realizados utilizando a mesma base de dados criada em [Ferreira et al. 2010b]. A base contém 1798 imagens extraídas do website Flickr[1], juntamente com as seguintes informações: *tags*, uma pequena descrição, a cidade com sua latitude e longitude e a imagem. São imagens das 14 maravilhas, 7 do antigo e 7 do novo mundo, mais o complexo das pirâmides de Gizé, a maravilha remanescente do mundo antigo. Existem 100 imagens de cada uma das 15 maravilhas, recuperadas juntamente com as descrições que as pessoas em geral publicam no website.

Das 1798 imagens, 1500 têm o atributo `Treino` com o valor `'Falso'`. As 298 imagens restantes, têm `Treino='True'` e, portanto, foram submetidas ao treinamento do MMD. Elas correspondem a exemplos do tipo de imagens que os usuários esperam obter de cada maravilha. É garantido que na base há pelo menos 20 imagens de cada uma das 15 maravilhas que atendem às expectativas do usuário.

Regras. Ao submetermos o comando `CREATE MINING MODEL ModeloMaravilhasDoMundo`, como no Exemplo 6.5, foram geradas 1799 regras de associação. Aplicando o algoritmo de tradução que propusemos neste estudo de caso, obtivemos 6352 regras de preferência condicional para o modelo `PrefModeloMaravilhasDoMundo`.

Consultas. Quatro consultas kNN com centros de consulta distintos foram utilizadas nos testes, com k assumindo os valores 1, 5, 10, 15 e 20. As imagens de consulta foram das maravilhas *Chichen Itza, Cristo Redentor, Complexo das Pirâmides de Gizé* e *Jardins*

[1] `http://www.flickr.com`

Suspensos da Babilônia, para Q1, Q2, Q3 e Q4, respectivamente. A Q1 está exemplificada abaixo:

```
SELECT ID, Image
FROM MaravilhasDoMundo
WHERE Image NEAR 'C:\ChichenItza.jpg' STOP AFTER k;
```

No caso da execução com o MP, as consultas assumiram $k = 50$ e o parâmetro n do MP é que variou com os valores 1, 5, 10, 15 e 20. Isto porque ao combinar operadores de similaridade com preferências, a semântica de uma consulta kNN é: primeiro selecione as k imagens mais similares à imagem de referência; depois, dentre elas, selecione até n imagens mais preferidas de acordo com o perfil do usuário. Então, aqui, escolhemos as 50 mais similares ao centro de consulta e depois, dentre essas 50, aplicamos o ***SelectK-Best*** para obter as n mais similares e preferidas, com $n \in \{1, 5, 10, 15, 20\}$.

Os gráficos da Figura 6.4 ilustram os resultados obtidos. Indiscutivelmente, o MMD com o modelo `ModeloMaravilhasDoMundo` aprimora as consultas por similaridade. Em todas as situações, sempre imagens relevantes foram obtidas, atingindo o máximo de precisão. Já o MP `PrefModeloMaravilhasDoMundo` não obteve bons resultados em consultas que recuperam poucas imagens. Para $n = 1$ e $n = 5$, percebemos que é melhor que o MP não esteja habilitado. Entretanto, à medida que a quantidade de imagens a serem devolvidas aumenta, o MP torna-se interessante. Na consulta Q1, por exemplo, dentre as 20 imagens mais similares, aproximadamente 50% são relevantes com o `PrefModeloMaravilhasDoMundo`. Sem ele, a precisão não chega a 30%.

6.2.4 Conclusões

Este estudo de caso é um complemento à implementação da linguagem CPref-SQL e um passo importante que abre possibilidades para um sistema mais completo, capaz de elicitar e executar consultas com preferências em bases que envolvem dados complexos.

Os resultados experimentais mostraram que ainda há muito o que ser feito. A precisão ainda está baixa e muito inferior àquela obtida pelo módulo de mineração de dados. Melhorias podem ser feitas, aprimorando a tradução das regras no passo de definição do parâmetro P para categorização ou mesmo nos critérios utilizados para garantir consistência.

Outro ponto importante deste estudo, é que ficou evidenciado que os algoritmos da CPref-SQL não escalam bem para grande quantidade de regras (maior que a própria base, neste caso). Em média, as consultas executaram em 180 segundos, sobre apenas 50 tuplas (imagens mais similares). Nos experimentos discutidos no Capítulo 5, o tempo médio de execução sobre uma base com 2333 tuplas, utilizando, entretanto, 40 regras, é de 100 segundos.

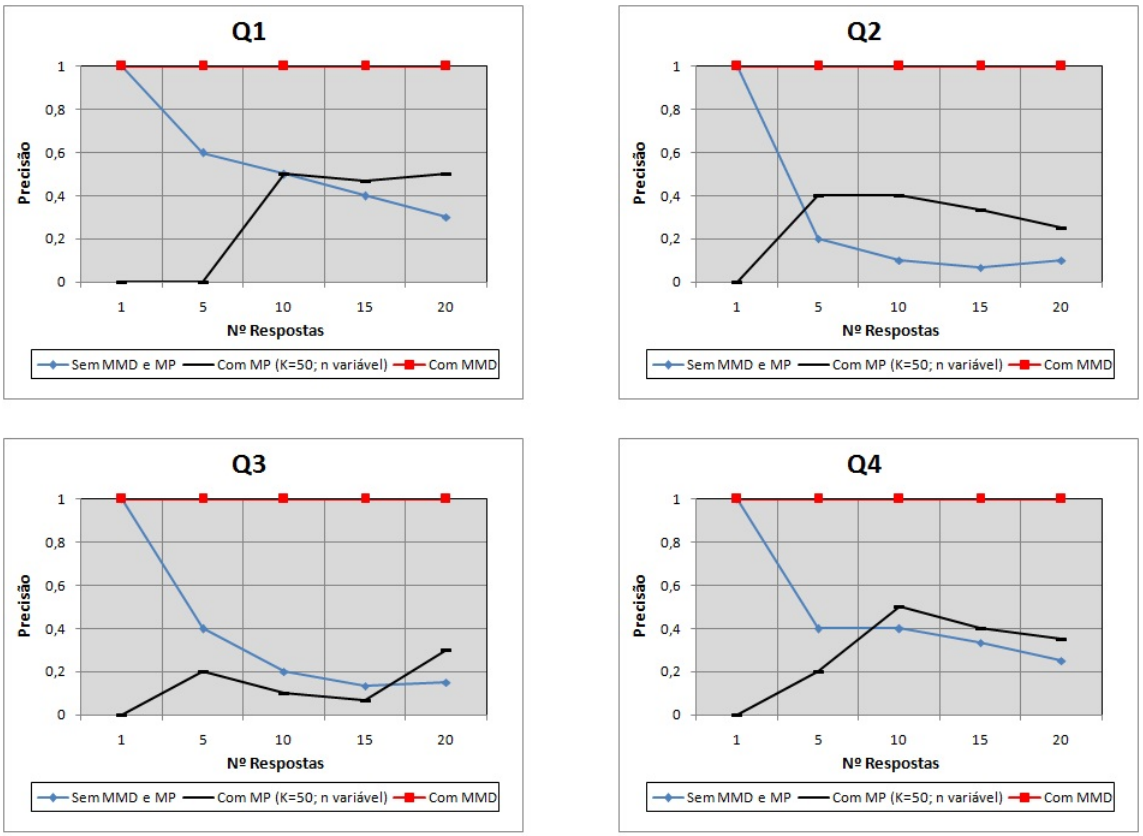


Figura 6.4: Precisão das respostas às consultas por similaridade

6.3 Considerações Finais

Os estudos de caso apresentados ilustram aplicações da linguagem CPref-SQL em combinação com o sistema de busca por similaridade SIREN. As propostas são resultado de um trabalho conjunto entre UFU e ICMC-USP pelo projeto PROCAD.

Primeiramente, desenvolvemos um módulo de preferência que permite que o SIREN realize consultas híbridas, contendo operadores de preferência e similaridade. O objetivo é poder recuperar as imagens mais similares e preferidas. Aplicamos essa abordagem no contexto de imagens médicas e mostramos, através de experimentos, que as preferências do médico aprimoram a qualidade das respostas.

Na segunda proposta, adaptamos o módulo de mineração de dados (MMD) do SIREN para elicitar automaticamente as preferências a serem utilizadas no módulo de preferência então implementado. É um complemento à CPref-SQL. A adaptação foi feita através da tradução de regras de associação obtidas pelo MMD em regras de preferência. Os resultados mostraram que o método é eficiente quando deseja-se recuperar uma maior quantidade de imagens.

Esses estudos são, portanto, exemplos das diversas aplicações que uma linguagem com suporte a preferências pode ter.

Capítulo 7

Conclusão

Preferências são onipresentes na vida real. Suponha que uma família deseja escolher um pacote de viagem para as férias no *site* de uma agência de turismo. O pai prefere visitar lugares históricos. Já os filhos preferem uma viagem para a praia, e a esposa tem preferência por viagens de curta duração. A escolha do melhor pacote turístico é um exemplo de situação que não poderia ser resolvida satisfazendo exatamente todos seus requisitos, mas sim através de preferências. Neste trabalho implementamos a CPref-SQL: uma linguagem de consulta com suporte a preferências condicionais.

CPref-SQL é uma extensão SQL que permite expressar preferências condicionais sob a abordagem qualitativa. As consultas em CPref-SQL incorporam as usuais restrições *hard* (declaradas na cláusula `WHERE`), além das restrições *soft*, especificadas através de um conjunto de regras de preferências e referenciadas pela cláusula `ACCORDING TO PREFERENCES`.

Através da CPref-SQL é possível realizar consultas top-k que retornam exatamente as K tuplas mais preferidas pelo usuário. A semântica e especificação do conjunto de regras de preferência são dadas pelo modelo de preferência que desenvolvemos, baseado no formalismo de teorias de preferências condicionais do trabalho de [Wilson 2004]. A implementação foi feita diretamente no código do processador de consultas do SGBDR PostgreSQL.

Para resolver, então, o problema acima, a CPref-SQL poderia ser incorporada ao *site* da agência de turismo, e, ao invés de retornar um resultado vazio, o *site* poderia oferecer algumas opções com roteiro histórico, outras para a praia, sempre buscando pequenas durações. É, portanto, uma linguagem que presta uma importante contribuição para a área de banco de dados, permitindo que essa nova modalidade de consultas — com preferências — possa ser especificada e avaliada.

7.1 Principais Contribuições

Para atingirmos o objetivo final de realizar consultas em um SGBDR utilizando a CPref-SQL, este trabalho de pesquisa passou por três etapas principais: (1) extensão do

modelo de preferência; (2) proposta de um operador para consultas top-k e algoritmos que resolvem os operadores de preferência da linguagem e (3) codificação dos algoritmos propostos. As principais contribuições desta dissertação são sintetizadas nos seguintes itens.

- Estendemos o modelo de preferência da CPref-SQL com o intuito de tornar a linguagem mais expressiva. Agora é possível expressar predicados mais genéricos nas regras de preferência, tais como $>$, $<$, $\leq$, entre outros. Além disso, inserimos uma nova ordem de preferência no modelo, a ordem fraca, oferecendo como alternativa uma semântica menos conservativa à linguagem.
- Com um modelo de preferência genérico, propusemos o conceito de consultas-pc top-k através do operador ***SelectK-Best*** para a CPref-SQL. A partir dos dois operadores de preferência bem definidos, desenvolvemos os algoritmos **G-BNL** e **GRank-BNL** que os implementam. Também propusemos algoritmos que implementam o teste de dominância para os dois tipos de ordem de preferência do modelo: **strongDomTest** para ordem forte e **weakDomTest** para ordem fraca.
- Por fim, codificamos a linguagem CPref-SQL sob a abordagem *built-in*, ou seja, diretamente no processador de consultas do SGBDR PostgreSQL. Como resultado, apresentamos duas extensões do Postgres com suporte à CPref-SQL: uma que utiliza a ordem forte de dominância e a outra, a ordem fraca. Nessas implementações, a regras-pc podem ser criadas com os predicados $=$, $\neq$, $>$, $<$, $\leq$ e $\geq$. Os resultados experimentais mostraram que, em geral, a ordem forte, apesar de ser menos relaxada, é mais eficiente. E ainda, que as implementações da CPref-SQL têm um desempenho superior à simples tradução de consultas em SQL com recursão. O ideal, então, é que o usuário avalie questões de performance e semântica, e escolha a abordagem CPref-SQL mais adequada às suas necessidades.
- Para ilustrar os resultados obtidos, propusemos duas aplicações da CPref-SQL no contexto de combinação com operadores de similaridade.

As contribuições deste trabalho de mestrado resultaram em 3 trabalhos publicados e 1 artigo a ser submetido a uma revista internacional.

1. As ideias iniciais de pesquisa, propostas e andamento do trabalho foram discutidas no artigo [Pereira and de Amo 2010b], apresentado no *IX Workshop de Teses e Dissertações em Banco de Dados*.
2. Em [Pereira and de Amo 2010a] é apresentada a primeira versão da implementação da CPref-SQL. Esse primeiro protótipo leva em conta o modelo de preferência sem extensões, ou seja, apenas com o predicado de igualdade ($=$), cujo teste de dominância é feito sob a ordem de preferência forte. É uma publicação na revista *Journal*

of Information and Data Management, que foi apresentada no *XXV Simpósio Brasileiro de Banco de Dados* e eleita uma das cinco melhores do evento.

3. A aplicação da CPref-SQL através do módulo de preferência inserido no SIREN resultou na publicação [Ferreira et al. 2010a], apresentada no simpósio *Computer-Based Medical Systems*.

4. A descrição detalhada do modelo de preferência generalizado, bem como as implementações com os dois tipos de teste de dominância e uma proposta para incorporar contextos na linguagem CPref-SQL estão no artigo *"A Context-Aware Preference Query Language: Theory and Implementation"*, por S. Amo e F. S. F. Pereira, que está em preparação e será submetido para a revista *Knowledge and Data Engineering*.

7.2 Propostas para Trabalhos Futuros

Com a implementação da CPref-SQL, vislumbramos diversas otimizações, extensões e aplicações do tema, que podem proporcionar pesquisas futuras.

1. Otimizações na implementação da linguagem CPref-SQL.

   - A implementação atual da CPref-SQL foi feita apenas para o bloco básico `SELECT ... FROM ... WHERE ...`. Naturalmente, uma primeira melhoria é incrementar a implementação da linguagem para suporte às demais cláusulas SQL: `ORDER BY`, agregados, etc.
   - Os algoritmos que propusemos para resolver os operadores de preferência ainda possuem um alto grau de complexidade. Com certeza é possível otimizar a avaliação dos operadores, diminuindo complexidades. Uma proposta, por exemplo, é fazer com que o cálculo do fecho transitivo dos algoritmos para teste de dominância seja feito apenas uma vez, no momento da criação das preferências. Dessa maneira, o teste de dominância nas consultas ficará com complexidade linear no número de regras obtidas.
   - Como a atual implementação está *built-in*, e com novas opções de algoritmos, é possível aproveitar o otimizador do sistema para a escolha do melhor plano de consulta.
   - Um outro trabalho interessante é a implementação da linguagem sob a abordagem *on-top*, oferecendo alternativa para execução de consultas com preferências sobre diferentes versões e SGBDRs. O protótipo com uma versão mais simples da CPref-SQL, implementado independente do código-fonte do SGBDR, que descrevemos no Capítulo 4, é uma prova de que é possível utilizar tal abordagem.

2. Tendências de pesquisa em preferências que podem ser incorporadas à CPref-SQL.

   - Preferências contextualizadas. A Figura 7.1 ilustra como o nível de exigência sobre a informação a ser recuperada tem aumentado e ganhado novos fatores. A ideia agora é inserir recursos como contexto e incertezas sobre as consultas, personalizando ainda mais os resultados. Os trabalhos de [Stefanidis et al. 2007] e [Levandoski et al. 2010a] são exemplos de como combinar os temas preferências e contexto. Uma proposta de estudo é inserir na CPref-SQL o recurso de avaliar consultas-pc baseadas em contexto, ou seja, as preferências de um usuário podem variar de acordo com as circunstâncias que se encontra. Por exemplo, se ele estiver procurando por uma opção de viagem com a família, é melhor que sejam roteiros históricos. Se for viajar com amigos, então a preferência é por opções para fazer um cruzeiro.

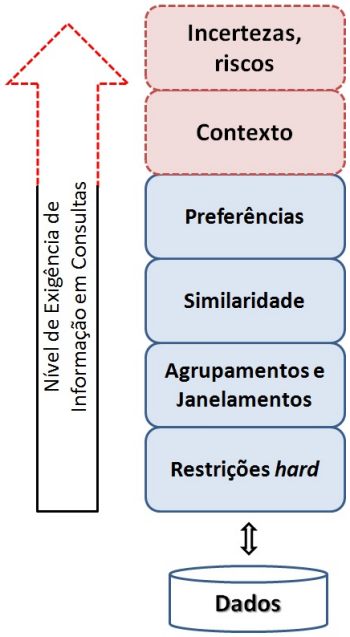


Figura 7.1: Novas tendências de consultas em SGBDs

   - Preferências sobre estruturas. As pesquisas discutidas em [de Amo and Giacometti 2008, Zhang and Chomicki 2011] estão voltadas para o fato de que pode ser interessante trabalhar com preferências não só sobre objetos simples, mas sim sobre estruturas, como por exemplo, conjuntos ou sequências. Um estudo importante, então, é incorporar à CPref-SQL funcionalidades que possibilite a execução de consultas com preferências sobre diversos tipos de estruturas, não apenas sobre tuplas individualmente.
   - Preferências em fluxos de dados (*data streams*). Atualmente, um número significante de aplicações requer a manipulação de dados que variam rapidamente em função do tempo, por exemplo, dados de sensores. [Kontaki 2010] é um

exemplo de trabalho que correlaciona o tema de preferências a esses tipos de dados. Dessa maneira, algoritmos que avaliam *data streams* com preferências podem ser desenvolvidos e incorporados à linguagem CPref-SQL, aproveitando sua sintaxe e modelo de preferência.

3. Aplicações da linguagem de preferências.

   - A necessidade de abordar preferências está inserida até na área de redes. Em [Mouratidis 2010], a proposta é utilizar consultas com preferências para avaliar o custo das rotas dos pacotes. As preferências são importantes porque existem diferentes fatores e formas para calcular custos, como distância Euclidiana, tempo de direção, tempo de caminhamento, etc. Logo, a CPref-SQL pode ser útil nas buscas realizadas pelos sistemas de rede.
   - Por fim, o trabalho [da Silva and de Amo 2011] trata de um importante assunto na área: elicitação de preferências. Foi construído um minerador de preferências que obtém preferências condicionais a partir de um conjunto de amostras fornecidas pelo usuário. Como trabalho de futuro e exemplo de aplicação da CPref-SQL, os dois trabalhos podem ser combinados, resultando num sistema integrado ao SGBDR que fornece desde um método de obtenção de preferências, até sua avaliação. É uma alternativa ao método que avaliamos no Capítulo 6 que utiliza o minerador do SIREN para elicitar preferências.

Referências Bibliográficas

[Agrawal and Srikant 1994] Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *20th International Conference on Very Large Data Bases, VLDB*, pages 487–499, Chile.

[Agrawal and Wimmers 2000] Agrawal, R. and Wimmers, E. (2000). A framework for expressing and combining preferences. *SIGMOD Rec.*, 29:297–306.

[Barioni et al. 2006] Barioni, M. C. N., Razente, H., Traina, A., and Traina, Jr., C. (2006). Siren: a similarity retrieval engine for complex data. In *Proceedings of the 32nd international conference on Very large data bases*, pages 1155–1158.

[Börzsönyi et al. 2001] Börzsönyi, S., Kossmann, D., and Stocker, K. (2001). The skyline operator. In *17th International Conference on Data Engineering (ICDE)*, pages 421–430.

[Boutilier et al. 2004] Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., and Poole, D. (2004). Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research (JAIR)*, 21:135–191.

[Brafman and Domshlak 2002] Brafman, R. I. and Domshlak, C. (2002). Introducing variable importance tradeoffs into cp-nets. In *18th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 69–76.

[Brafman et al. 2006] Brafman, R. I., Domshlak, C., and Shimony, S. E. (2006). On graphical modeling of preference and importance. *Journal of Artificial Intelligence Research (JAIR)*, 25:389–424.

[B.W. Silverman and Hodges 1989] B.W. Silverman, M.C. Jones, E. F. and Hodges, J. (1989). An important contribution to nonparametric discriminant analysis and density estimation–commentary on fix and hodges. *International Statistical Review*, 57:233–247.

[Carey and Kossman 1997] Carey, M. and Kossman, D. (1997). On saying "enough already"in sql. *SIGMOD Record*, 26(2):219–230.

[Chan et al. 2005a] Chan, C.-Y., Eng, P.-K., and Tan, K.-L. (2005a). Efficient processing of skyline queries with partially-ordered domains. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, pages 190–191, Washington, USA.

[Chan et al. 2005b] Chan, C.-Y., Eng, P.-K., and Tan, K.-L. (2005b). Stratified computation of skylines with partially-ordered domains. In *Proceedings of the 2005 ACM*

SIGMOD international conference on Management of data, pages 203–214, New York, USA.

[Chomicki 2003] Chomicki, J. (2003). Preference formulas in relational queries. *ACM Transactions on Database Systems*, 28(4):427–466.

[Conway and Sherry 2006] Conway, N. and Sherry, G. (2006). Introduction to hacking postgresql.

[da Silva and de Amo 2011] da Silva, N. F. F. and de Amo, S. (2011). Cprefminer: A bayesian miner of conditional preferences. *Journal of Information and Data Management (JIDM)*.

[de Amo and Giacometti 2008] de Amo, S. and Giacometti, A. (2008). *Preferences on Objects, Sets and Sequences*. I-Tech Publications, Vienna, Austria.

[de Amo and Ribeiro 2009] de Amo, S. and Ribeiro, M. R. (2009). Cpref-sql: A query language supporting conditional preferences. In *24th Annual ACM Symposium on Applied Computing (ACM SAC 2009)*.

[Eder 2009] Eder, H. (2009). *On Extending PostgreSQL with the Skyline Operator*. PhD thesis, Vienna University of Technology.

[Eder and Wei 2009] Eder, H. and Wei, F. (2009). Evaluation of skyline algorithms in postgresql. In *13th International Database Engineering & Application Symposium (IDEAS 09)*.

[Ferreira et al. 2010a] Ferreira, M. R. P., Pereira, F. S. F., Ponciano-Silva, M., de Amo, S., Traina, A. J. M., Traina, Jr., C., and Chbeir, R. (2010a). Integrating user preference to similarity queries over medical images datasets. In *23rd IEEE International Symposium on Computer-Based Medical Systems (CBMS 2010)*, pages 486–491.

[Ferreira et al. 2010b] Ferreira, M. R. P., Ribeiro, M. X., Traina, A. J. M., Chbeir, R., and Traina, Jr., C. (2010b). Adding knowledge extracted by association rules into similarity queries. In *Journal of Information and Data Management (JIDM)*, volume 1, pages 391–406.

[Ferreira et al. 2009] Ferreira, M. R. P., Traina, A. J. M., Dias, I., Chbeir, R., and Traina, Jr., C. (2009). Identifying algebraic properties to support optimization of unary similarity queries. In *3rd Alberto Mendelzon International Workshop on Foundations of Data Management*, pages 1–10, Peru.

[Goldsmith et al. 2008] Goldsmith, J., Lang, J., Truszczynski, M., and Wilson, N. (2008). The computational complexity of dominance and consistency in cp-nets. *J. Artificial Intelligence Res.*, 33:403–432.

[Hristidis et al. 2001] Hristidis, V., Koudas, N., and Papakonstantinou, Y. (2001). Prefer: A system for the efficient execution of multi-parametric ranked queries. In *Proceedings of the ACM SIGMOD Conf.*, pages 259–270, Santa Barbara, USA.

[ISO 1992] ISO (1992). Database language sql isoiec 9075 : 1992.

[ISO 1999] ISO (1999). Database language sql isoiec 9075 – 2 : 1999.

[Kießling 2002] Kießling, W. (2002). Foundations of preferences in database systems. In *28th International Conference International Conference on Very Large Data Bases (VLDB)*, pages 311–322.

[Kießling and Köstler 2002] Kießling, W. and Köstler, G. (2002). Preference sql - design, implementation, experiences. In *28th International Conference International Conference on Very Large Data Bases (VLDB)*, pages 990–1001.

[Kim et al. 2010] Kim, R., Dasovich, G., Bhaumik, R., Brock, R., Furst, J. D., and Raicu, D. S. (2010). An investigation into the relationship between semantic and content based similarity using lidc. In *International Conference on Multimedia Information Retrieval (MIR'10)*, pages 185–192, USA.

[Kontaki 2010] Kontaki, M., P. A. M. Y. (2010). Continuous processing of preference queries in data streams. In *36th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 47–60.

[Koutrika and Ioannidis 2005] Koutrika, G. and Ioannidis, Y. (2005). Personalized queries under a generalized preference model. In *21st International Conference On Data Engineering (ICDE)*, pages 841–852, Japan.

[Koutrika et al. 2011] Koutrika, G., Pitoura, E., and Stefanidis, K. (2011). A survey on representation, composition and application of preferences in database system. *ACM Transactions on Database Systems*, 36(4).

[Koutrika et al. 2006] Koutrika, G., Simitsis, A., and Ionnadis, Y. (2006). Précis: The essence of a query answer. In *Proceedings of the 22nd International Conference on Data Engineering*, pages 69–78, Atlanta, USA.

[Levandoski et al. 2010a] Levandoski, J. J., Mokbel, M. F., and Khalefa, M. E. (2010a). Caredb: a context and preference-aware location-based database system. *Proc. VLDB Endow.*, 3:1529–1532.

[Levandoski et al. 2010b] Levandoski, J. J., Mokbel, M. F., and Khalefa, M. E. (2010b). Flexpref: A framework for extensible preference evaluation in database systems. In *26th International Conference on Data Engineering (ICDE)*, pages 828–839 .

[Levandoski et al. 2010c] Levandoski, J. J., Mokbel, M. F., Khalefa, M. E., and Korukanti, V. (2010c). A demonstration of flexpref: extensible preference evaluation inside the dbms engine. In *Proceedings of the 2010 international conference on Management of data*, pages 1247–1250.

[Lin et al. 2007] Lin, X., Yuan, Y., Zhang, Q., and Zhang, Y. (2007). Selecting stars: The k most representative skyline operator. In *23th International Conference on Data Engineering (ICDE)*, pages 86 – 95.

[Lloyd 1984] Lloyd, J. (1984). *Foundations of Logic Programming*. Springer Verlag.

[Ludovic et al. 2007] Ludovic, L., Daniel, R., and Tbahriti, S.-E. (2007). Towards an extended sqlf: Bipolar query language with preferences. *World Academy of Science, Engineering and Technology*, pages 305–310.

[Mouratidis 2010] Mouratidis, K., L. Y. Y. M. (2010). Preference queries in large multi-cost transportation networks. In *IEEE International Conference on Data Engineering (ICDE)*, pages 533–544.

[Papadias et al. 2005] Papadias, D., Tao, Y., Fu, G., and Seeger, B. (2005). Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82.

[Pereira and de Amo 2010a] Pereira, F. S. F. and de Amo, S. (2010a). Evaluation of conditional preference queries. In *Journal of Information and Data Management*, volume 1.

[Pereira and de Amo 2010b] Pereira, F. S. F. and de Amo, S. (2010b). Implementação de uma linguagem de consulta com suporte a preferências condicionais. In *XXV Simpósio Brasileiro de Banco de Dados - Workshop Teses e Dissertações*, Belo Horizonte.

[Ponciano-Silva et al. 2009] Ponciano-Silva, M., Traina, A. J. M., Azevedo-Marques, P. M., Felipe, J. C., and Traina, Jr., C. (2009). Including the perceptual parameter to tune the retrieval ability of pulmonary cbir systems. In *22nd IEEE International Symposium on Computer-Based Medical Systems*, pages 1–8, USA.

[Preisinger and Kießling 2007] Preisinger, T. and Kießling, W. (2007). The hexagon algorithm for pareto preference queries. In *Proceedings of the 3rd Multidisciplinary Workshop on Advances in Preference Handling, VLDB*, Vienna, Austria.

[Preisinger et al. 2006] Preisinger, T., Kießling, W., and Endres, M. (2006). The bnl++ algorithm for evaluating pareto preference queries. In *2nd Multidisciplinary Workshop on Advances in Preference Handling (MPREF)*, pages 114–121.

[Ribeiro 2008] Ribeiro, M. R. (2008). Linguagens de consulta para banco de dados com suporte a preferências condicionais. Master's thesis, Universidade Federal de Uberlândia.

[Ribeiro et al. 2008] Ribeiro, M. X., Ferreira, M. R. P., Traina, Jr., C., and Traina, A. J. M. (2008). Data pre-processing: a new algorithm for feature selection and data discretization. In *5th International Conference on Soft Computing as Transdisciplinary Science and Technology (CSTST 2008)*, pages 252–257.

[Rorissa et al. 2008] Rorissa, A., Clough, P., and Deselaers, T. (2008). Exploring the relationship between feature and perceptual visual spaces. *Journal of the American Society for Information Science and Technology (JASIST)*, 59:770–784.

[Siberski et al. 2006] Siberski, W., Pan, J. Z., and Thaden, U. (2006). Querying the semantic web with preferences. In *The 5th International Semantic Web Conference (ISWC)*, pages 612–624.

[Stefanidis et al. 2007] Stefanidis, K., Pitoura, E., and Vassiliadis, P. (2007). Adding context to preferences. In *23rd International Conference on Data Engineering (ICDE)*, pages 846–855.

[TPC 2010] TPC (2010). Benchmark h (decision support) standard specification. revision 2.10.0.

[Wilson 2004] Wilson, N. (2004). Extending cp-nets with stronger conditional preference statements. In *19th National Conference on Artificial Intelligence (AAAI)*, pages 735–741.

[Yiu and Mamoulis 2007] Yiu, M. L. and Mamoulis, N. (2007). Efficient processing of top-k dominating queries on multi-dimensional data. In *Proceedings of the 33rd international Conference on Very large Data Bases*, pages 483–494, Vienna, Austria.

[Yiu and Mamoulis 2009] Yiu, M. L. and Mamoulis, N. (2009). Multi-dimensional top-k dominating queries. *The VLDB Journal*, 18(3):695–718.

[Zhang and Chomicki 2011] Zhang, X. and Chomicki, J. (2011). Preference queries over sets. In *27th International Conference on Data Engineering (ICDE)*.

[Zhou et al. 2008] Zhou, X. S., Zillner, S., Moeller, M., Sintek, M., Zhan, Y., Krishnan, A., and Gupta, A. (2008). Semantics and cbir: a medical imaging perspective. In *Proceedings of the 2008 international conference on Content-based image and video retrieval*, pages 571–580, USA.

Apêndice A

A Linguagem CPref-SQL na Web

Para fins de organização e divulgação do trabalho que desenvolvemos, construímos um *site* para a linguagem CPref-SQL: `http://www.lsi.ufu.br/cprefsql`.

Nele, concentramos informações acerca das publicações, documentação e *download* dos códigos desenvolvidos. A Figura A.1 ilustra a página inicial do *site*.

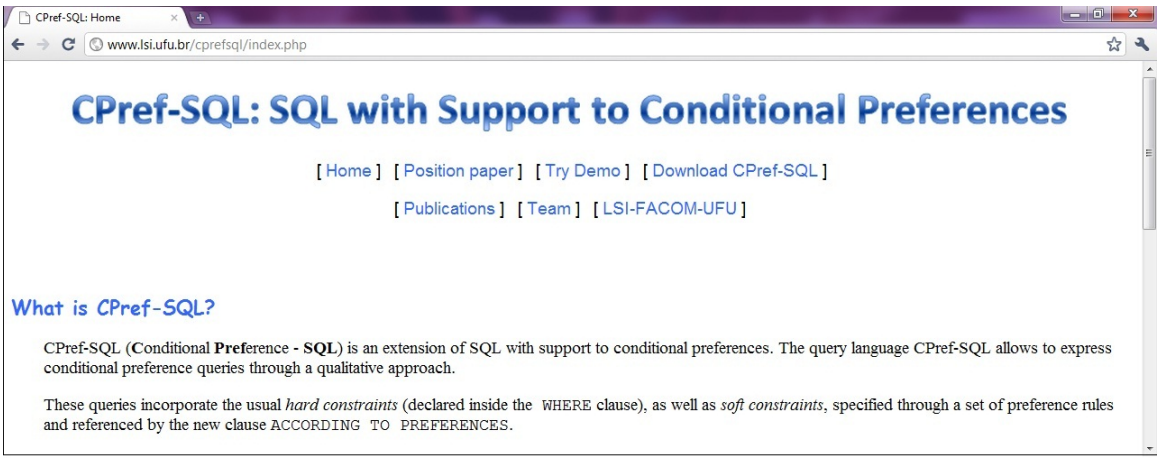


Figura A.1: Página inicial do *site*

Em especial, uma das principais funcionalidades desse *site* é a interface de demonstração da linguagem, na qual é possível realizar consultas em CPref-SQL sobre pequenas relações que disponibilizamos como exemplo. A Figura A.2 é um *snapshot* da página de demonstração.

A dinâmica de execução de consultas é a seguinte:

- Preferências podem ser criadas sobre três relações existentes no banco de dados: *movies*, *travels* ou *hotels*.
- A instância existente dessas relações pode ser visualizada clicando nos respectivos botões do lado direito da tela.
- Para criar preferências, basta executar o comando `CREATE PREFERENCES` no campo de texto (Figura A.3). Para efeito de organização, toda preferência criada deve

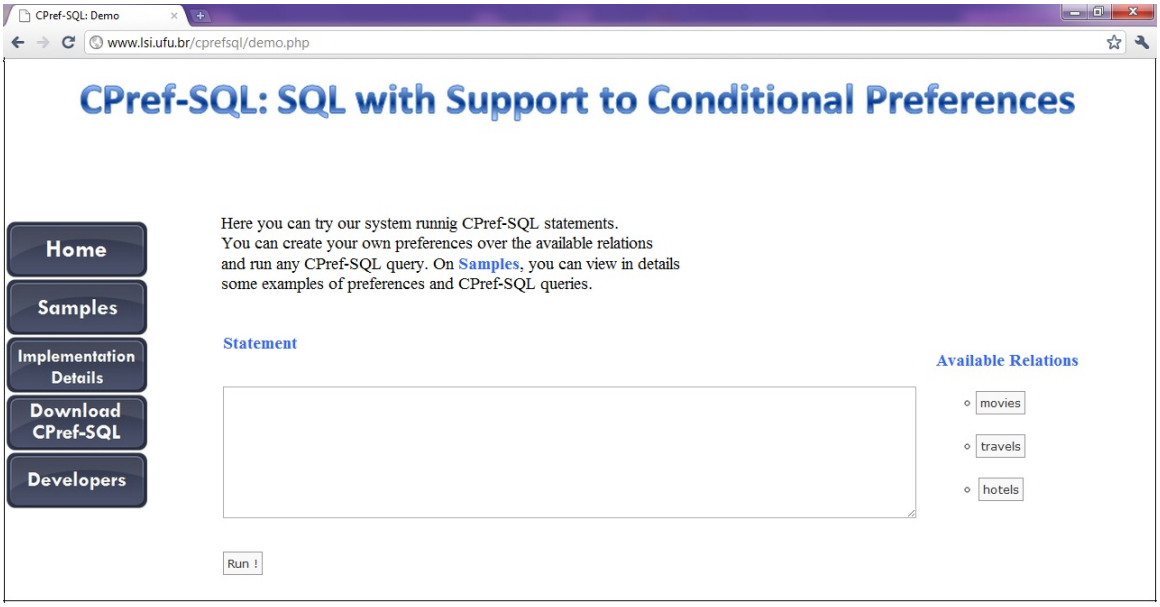


Figura A.2: Interface de demonstração

ser denominada *mypref* de modo que, a cada criação, as preferências antigas são sobrescritas.

- Uma vez criadas as preferências, agora é possível realizar consultas utilizando a cláusula `ACCORDING TO PREFERENCES` no comando `SELECT` (Figura A.4).

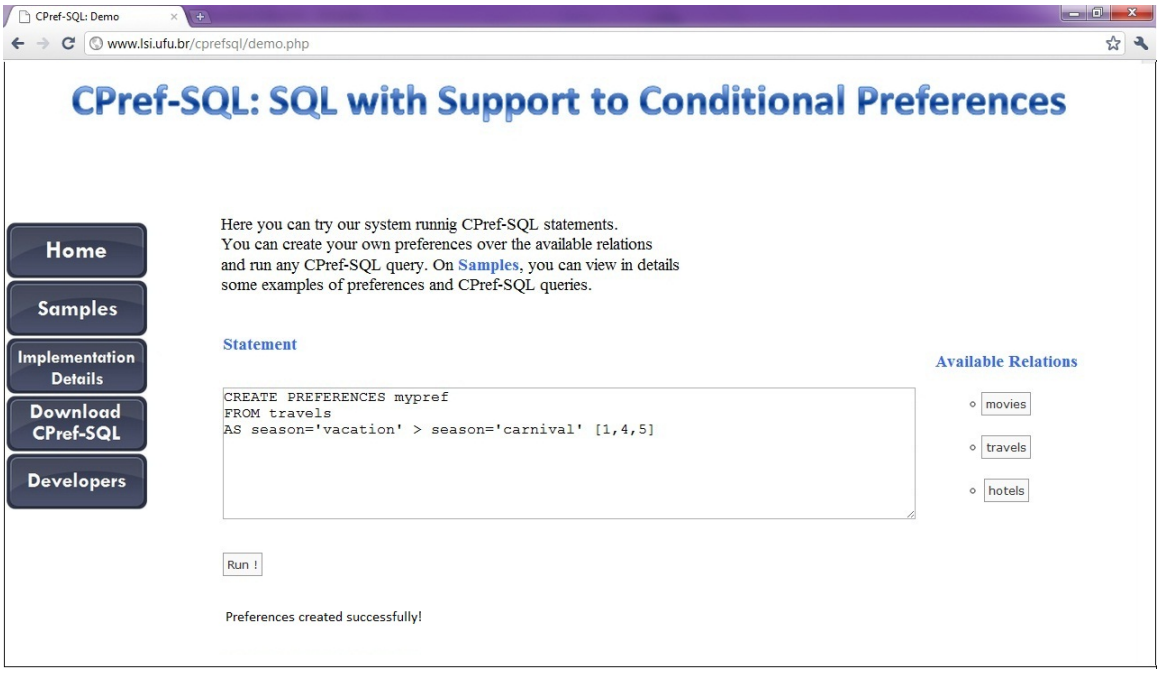


Figura A.3: Interface de demonstração - `CREATE PREFERENCES`

Vale ressaltar que, nessa interface, as consultas CPref-SQL são executadas sobre a implementação *built-in*, com a abordagem de ordem forte de preferência.

Na página *Samples* disponibilizamos diversos exemplos que podem ser replicados na interface de consultas para ilustrar o funcionamento da linguagem CPref-SQL.

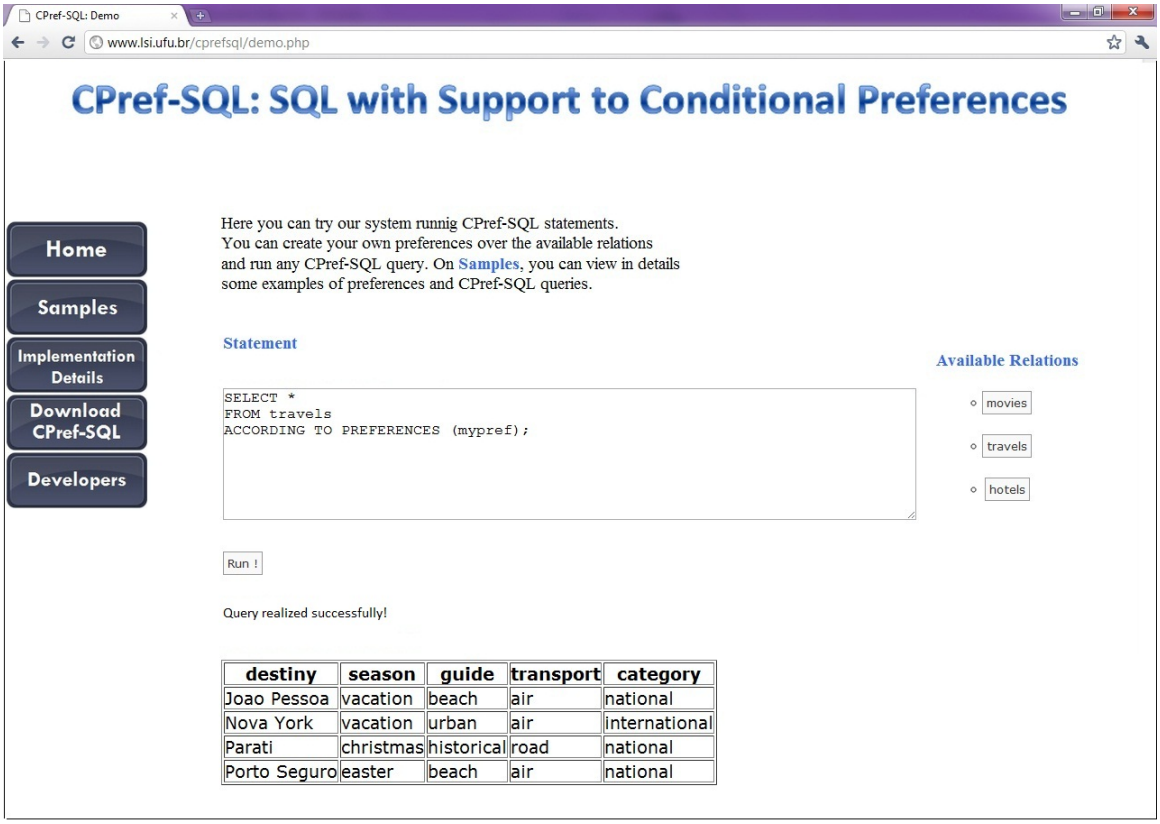


Figura A.4: Interface de demonstração - bloco `SELECT`

O desenvolvimento deste *site* é, portanto, uma contribuição em termos de organização e melhor entendimento da linguagem CPref-SQL. Através das informações disponíveis, acreditamos facilitar e motivar futuros trabalhos que envolvam este tema.