Banco de Dados Dedutivo

Daniela Costa
Ítalo Francyles
Sidney Melo
Thacyla Lima

Bancos de dados dedutivos

- Os bancos de dados dedutivos, além de armazenar informações explícitas, permitem definir regras

- Possibilita que inferências sejam feitas a partir de fatos já armazenados no banco de dados

Bancos de dados dedutivos

Fatos

- especificados por meio de uma linguagem declarativa
- modo semelhante à forma como as relações são especificadas
- não é incluso o nome do atributo
- valor de atributo em uma tupla é determinado por sua posição dentro da tupla
- mecanismo de inferência pode deduzir novos fatos do banco de dados ao interpretar regras

Regras

- especificam relações virtuais que não estão armazenadas, mas podem ser formadas com base nos fatos

- podem envolver recursão e gerar relações virtuais que não podem ser definidas em relação a visões relacionais básicas.

Bancos de dados dedutivos

- Prolog é uma linguagem tipicamente usada para especificar fatos, regras e consultas em bancos de dados dedutivos.

- Datalog, uma variação da linguagem Prolog, é usada para definir regras em forma de declaração, junto com um conjunto de relações existentes, que por si só são tratadas como literais na linguagem.

Notação Prolog/Datalog

- Baseada em fornecer predicados com nomes exclusivos
- Um predicado tem um nome e um número fixo de argumentos
- Um fato é representado por um predicado com termos constantes
- Se o predicado tiver variáveis como argumentos, ele é considerado uma consulta ou parte de uma regra ou restrição

Notação Prolog/Datalog - Predicado

`SUPERVISIONA(X, Y)` declara o fato que X supervisiona Y

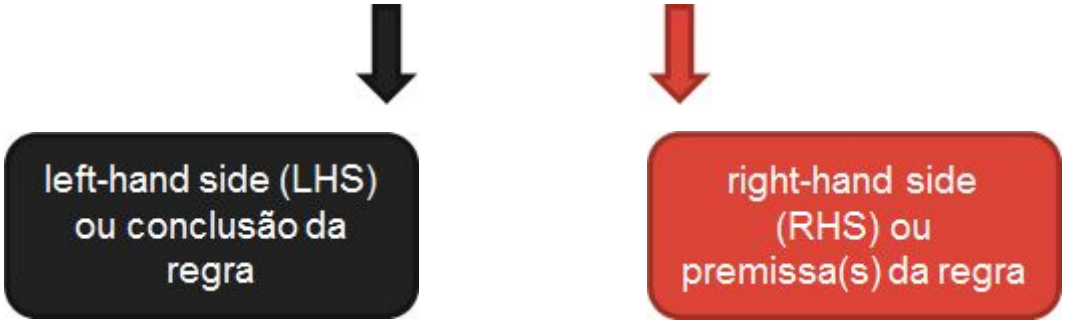
(a) Fatos

SUPERVISIONA (fernando, joao).
SUPERVISIONA (fernando, ronaldo).
SUPERVISIONA (fernando, joice).
SUPERVISIONA (jennifer, alice).
SUPERVISIONA (jennifer, andre).
SUPERVISIONA (jorge, fernando).
SUPERVISIONA (jorge, jennifer).
...

SUPERVISIONA(Supervisor, Supervisionado)

Notação Prolog/Datalog - Regra

cabeça :- corpo



Notação Prolog/Datalog

- Uma regra especifica que, se determinada atribuição dos valores constantes às variáveis no corpo tornar todos os predicados RHS verdadeiros, ele também torna a cabeça verdadeira ao usar a mesma atribuição de valores constantes às variáveis.

Regras

SUPERIOR(*X*, *Y*) :- SUPERVISIONA(*X*, *Y*).
SUPERIOR(*X*, *Y*) :- SUPERVISIONA(*X*, *Z*), SUPERIOR(*Z*, *Y*).
SUBORDINADO(*X*, *Y*) :- SUPERIOR(*Y*, *X*).

Notação Prolog/Datalog - Consulta

- Uma consulta envolve um símbolo de predicado com argumentos variáveis, e seu significado é deduzir todas as diferentes combinações de constantes que, quando vinculadas às variáveis, podem tornar o predicado verdadeiro

SUPERIOR(jorge, *Y*)?
SUPERIOR(jorge, joice)?

Notação Datalog

- Em Datalog, um programa é criado com base em objetos básicos, as fórmulas atômicas são literais na forma

$$p(a_1, a_2, \ldots a_n)$$

- Os predicados embutidos são de dois tipos principais: os de comparação binária < (less), <= (less_or_equal), > (greater) e >= (greater_or_equal) em domínios ordenados; e os predicados de comparação = e /= em domínios ordenados e não ordenados.

Notação Datalog

- Os programas Datalog podem ser considerados um subconjunto de fórmulas de cálculo de predicado.

- Essas fórmulas são primeiro convertidas em forma clausular antes que sejam expressas em Datalog, e somente fórmulas dadas em forma clausular restritas, denominadas cláusulas de Horn, podem ser usadas em Datalog.

Forma Clausal e Cláusula de horn

- As fórmulas do cálculo relacional incluem predicados e também os quantificadores universal $\forall$ e existencial $\exists$
- Em uma forma clausal, a fórmula precisa atender aos seguintes requisitos:
  - Todas as variáveis são universalmente quantificáveis
  - As cláusulas são disjunções de literais
  - A fórmula é uma conjunção de cláusulas

$$not(P_1)\ OR\ not(P_2)\ OR\ ..\ OR\ not(P_N)\ OR\ Q_1\ OR\ Q_2\ OR\ ...\ OR\ Q_N$$

com equivalente

$$P_1\ AND\ P_2\ AND\ ...\ AND\ P_n \Rightarrow Q_1\ OR\ Q_2\ OR\ ...\ OR\ Q_n$$

Forma Clausal e Cláusula de Horn

- Uma cláusula que contenha no máximo um literal positivo é chamada CLÁUSULA DE HORN
- Em Datalog, toda regra é uma cláusula de Horn.

$$P_1 \; AND \; P_2 \; AND \; ... \; AND \; P_n \Rightarrow Q_1 \; OR \; Q_2 \; OR \; ... \; OR \; Q_n$$

Em datalog:

$$Q :- \; P_1, P_2, ..., P_n$$

Q verdadeiro é uma restrição de integridade

Forma Clausal e Cláusula de horn

- Componentes de uma consulta datalog:
  - Um programa datalog com um conjunto finito de regras
  - Um conjunto de predicados $P(X_i)$, $m \in [1, m]$, onde cada X_i é variável constante
- Consulta Prolog X Datalog
  - Máquina de inferência
    - Prolog: um resultado de cada vez para cada consulta
    - Datalog: um conjunto de resultados por vez

Interpretação de Regras

- Abordagem prova-teórica
  - Os fatos são chamados de axiomas
  - aplica-se as regras aos fatos

superior(X,Y) :- supervisiona(X.Y)	(regra 1)
superior(X,Y) :- supervisiona(X.Z), superior(Z,Y)	(regra 2)
supervisiona(jennifer,ahmad)	(axioma fundamenta, dado)
supervisiona(james,jennifer)	(axioma fundamenta, dado)
superior(jennifer,ahmad)	(aplicar a regra 1 em 3)
superior(james,ahmad)	(aplicar a regra 2 em 4 e 5)

Interpretação de Regras

- Abordagem modelo-teórico
  - Informa-se um conjunto de predicados indicados como VERDADEIRO e FALSO
  - Os novos fatos verdadeiros são deduzidos através da combinação dos fatos verdadeiros já explicitados.

regra: superior(X.Y) : - supervisiona(X,Y).
superior(X.Y) : - supervisiona(X,Z), superior(Z.Y).

Fatos conhecidos	Fatos novos
supervisiona(jennifer, alicia) é verdadeiro.	superior(jennifer.alicia) é verdadeiro.
supervisiona(jennifer, jose) é verdadeiro.	superior(jennifer,jose) é verdadeiro.
supervisiona(james, jennifer) é verdadeiro.	superior(james, jose) é verdadeiro.

Programas Datolog e sua segurança

- **Predicados definidos por fatos (ou relações):** Correspondem a todas combinações de tuplas que tornam o predicado verdadeiro.
  Exemplo:

```
DEPARTAMENTO(jennifer, administracao).      TRABALHA_EM(joice, produtoy, 20).
DEPARTAMENTO(ronaldo, pesquisa).            TRABALHA_EM(fernando, produtoy, 10).
```

Figura --: Predicados definidos por fatos

- **Predicados definidos por regras (ou visões):** são definidos por serem a cabeça (LHS) de uma ou mais regras Datalog. Podem ser deduzidos pelo mecanismo de inferência.
  Exemplo:

```
SUPERIOR(X, Y) :- SUPERVISIONA(X, Y).
SUPERIOR(X, Y) :- SUPERVISIONA(X, Z), SUPERIOR(Z, Y).
```

Figura --: Predicados definidos por regras

Programas Datolog e sua segurança

- **Segurança**:
  - Um programa ou uma regra é considerado seguro se gerar um conjunto finito de fatos.
  - Uma regra é considerada segura se todas as suas **variáveis** forem **limitadas**.
  - Condições para que uma variável X seja limitada:
    - X aparece em um predicado regular no corpo da regra;
    - X é limitada por uma constante: X = c ou c = X ou (c1 <= X e X <= c2) no corpo da regra;
      Ou..
    - X é limitada por outra variável limitada: X = Y ou Y = X.

Exemplo de regra insegura: ALTO_SALARIO(Y) :- Y>6000 (y pode variar por todos os inteiros possíveis)
TEM_ALGO(X,Y) :- FUNCIONARIO(X)
Exemplo de regra segura: ALTO_SALARIO (Y) :- FUNCIONARIO(X), SALARIO(X,Y),Y>60000

Operações Relacionais

- Consultas e visões são facilmente especificadas em Datalog;
- A vantagem do Datalog sobre a álgebra relacional consiste no fato de que este permite consultas recursivas e visões baseadas em consultas recursivas;
- Dependendo do mecanismo de inferência, a eliminação automática de duplicatas pode ou não ocorrer. No entanto qualquer regra que envolva eliminação automática de duplicata estará incorreta em Prolog. Por exemplo, a operação de UNIÃO seria reescrita:
-

```
UNION_ONE_TWO(X, Y, Z) :– REL_ONE(X, Y, Z).
UNION_ONE_TWO(X, Y, Z) :– REL_TWO(X, Y, Z),
                          NOT(REL_ONE(X, Y, Z)).
```

Predicado em Datalog

```
REL_ONE(A, B, C).
REL_TWO(D, E, F).
REL_THREE(G, H, I, J).

SELECT_ONE_A_EQ_C(X, Y, Z) :- REL_ONE(C, Y, Z).
SELECT_ONE_B_LESS_5(X, Y, Z) :- REL_ONE(X, Y, Z), Y<5.
SELECT_ONE_A_EQ_C_AND_B_LESS_5(X, Y, Z) :- REL_ONE(C, Y, Z), Y<5.

SELECT_ONE_A_EQ_C_OR_B_LESS_5(X, Y, Z) :- REL_ONE(C, Y, Z).
SELECT_ONE_A_EQ_C_OR_B_LESS_5(X, Y, Z) :- REL_ONE(X, Y, Z), Y<5.

PROJECT_THREE_ON_G_H(W, X) :- REL_THREE(W, X, Y, Z).

UNION_ONE_TWO(X, Y, Z) :- REL_ONE(X, Y, Z).
UNION_ONE_TWO(X, Y, Z) :- REL_TWO(X, Y, Z).

INTERSECT_ONE_TWO(X, Y, Z) :- REL_ONE(X, Y, Z), :- REL_TWO(X, Y, Z).

DIFFERENCE_TWO_ONE(X, Y, Z) :- REL_TWO(X, Y, Z) NOT(REL_ONE(X, Y, Z).

CART PROD_ONE_THREE(T, U, V, W, X, Y, Z) :-
    REL_ONE(T, U, V), REL_THREE(W, X, Y, Z).

NATURAL_JOIN_ONE_THREE_C_EQ_G(U, V, W, X, Y, Z) :-
    REL_ONE(T, U, V), REL_THREE(W, X, Y, Z).
```

Predicados em Datalog

AVALIAÇÃO DE CONSULTAS DATALOG NÃO RECURSIVAS

- Neste momento veremos como um mecanismo de inferência baseado nas operações relacionais age para consultas Datalog não recursivas.
- Uma **consulta não recursiva** é caracterizada por ter apenas predicados não recursivos.
- 1ª situação: consulta que inclui apenas predicados definidos por fato => o processo de inferência consiste apenas em procurar o resultado da consulta nos fatos.
    - Exemplo: DEPARTAMENTO(X, Pesquisa)?
- 2ª situação : consulta que envolve predicados definidos por regra => o mecanismo de inferência precisa calcular primeiro as relações que estão no corpo da regra.
    - Exemplo: consulta que envolvam o predicado FUNC_ACIMA_40K(X) :- FUNCIONARIO(X), SALARIO(X,Y), Y >= 40000.
- É útil acompanhar a dependência entre os predicados de um banco de dados dedutivo em um ***Grafo de Dependência de Predicados.***

GRAFO DE DEPENDÊNCIA DE PREDICADOS

.

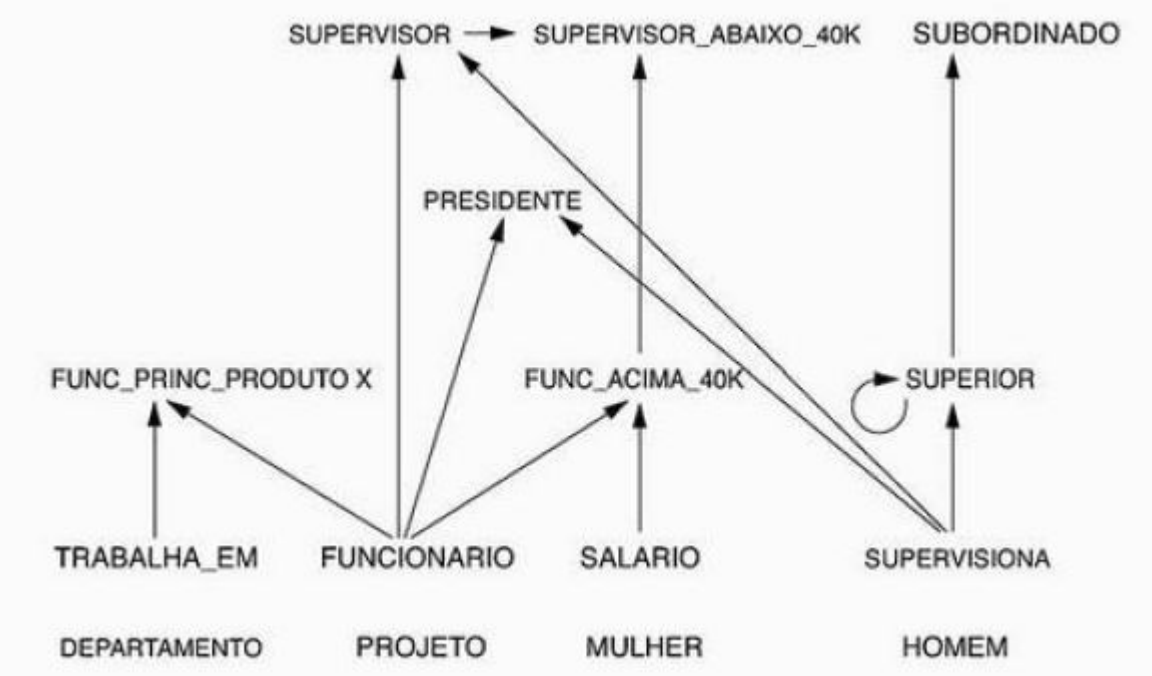


Figura --: Grafo de Dependência

Avaliação de Consultas Recursivas

- Regra não recursiva

  COMP (px, py) <= EP (px, py)

- Regra recursiva

  COMP (px, py) <= EP (px, py)

  COMP (px, py) <= COMP (px, pz) AND COMP (pz, py)

- Regra linearmente recursiva

  COMP (px, py) <= EP (px, py)

  COMP (px, py) <= EP (px, pz) AND COMP (pz, py)

Avaliação Ingênua

- Regra

  COMP (px, py) <= EP (px, py)

  COMP (px, py) <= EP (px, pz) AND COMP (pz, py)

- Algoritmo:

  COMP := EP;

  do until COMP chegar a um “ponto fixo”;

  COMP := COMP UNION (COMP # EP);

  end;

  DISPLAY := COMP WHERE PX = P#(‘P1’);

EP	
PX	PY
P1	P2
P1	P3
P2	P3
P2	P4
P3	P5
P4	P5
P5	P6

Avaliação Ingênua

COMP	
PX	PY
P1	P2
P1	P3
P2	P3
P2	P4
P3	P5
P4	P5
P5	P6

EP	
PX	PY
P1	P2
P1	P3
P2	P3
P2	P4
P3	P5
P4	P5
P5	P6

COMP # EP	
PX	PY
P1	P3
P1	P4
P1	P5
P2	P5
P3	P6
P4	P6

COMP	
PX	PY
P1	P2
P1	P3
P2	P3
P2	P4
P3	P5
P4	P5
P5	P6
P1	**P4**
P1	**P5**
P2	**P5**
P3	**P6**
P4	**P6**

COMP	
PX	PY
P1	P2
P1	P3
P2	P3
P2	P4
P3	P5
P4	P5
P5	P6
P1	**P4**
P1	**P5**
P2	**P5**
P3	**P6**
P4	**P6**

EP	
PX	PY
P1	P2
P1	P3
P2	P3
P2	P4
P3	P5
P4	P5
P5	P6

COMP # EP	
PX	PY
P1	P3
P1	P4
P1	P5
P2	P5
P3	P6
P4	P6
P1	P6
P2	P6

COMP	
PX	PY
P1	P2
P1	P3
P2	P3
P2	P4
P3	P5
P4	P5
P5	P6
P1	P4
P1	P5
P2	P5
P3	P6
P4	P6
P1	**P6**
P2	**P6**

Avaliação Ingênua

- Regra

  COMP (px, py) <= EP (px, py)

  COMP (px, py) <= EP (px, pz) AND COMP (pz, py)

- Algoritmo:

  COMP := EP;

  do until COMP chegar a um “ponto fixo”;

  COMP := COMP UNION (COMP # EP);

  end;

  DISPLAY := COMP WHERE PX = P#(‘P1’);

- Conclusão
  - Repetitivo e ineficiente

DISPLAY	
PX	PY
P1	P2
P1	P3
P1	P4
P1	P5
P1	P6

Avaliação Semi-Ingênua

- Algoritmo:
  NOVA = EP;
  COMP := NOVA;
  do until NOVA estar vazia;
  NOVA := (NOVA # EP) MINUS COMP;
  COMP := COMP UNION NOVA;
  end;
  DISPLAY := COMP WHERE PX = P#('P1');

- Evita repetição de cálculos

EP	
PX	PY
P1	P2
P1	P3
P2	P3
P2	P4
P3	P5
P4	P5
P5	P6

Avaliação Semi-Ingênua

NOVA	
PX	PY
P1	P2
P1	P3
P2	P3
P2	P4
P3	P5
P4	P5
P5	P6

COMP	
PX	PY
P1	P2
P1	P3
P2	P3
P2	P4
P3	P5
P4	P5
P5	P6

NOVA	
PX	PY
P1	P4
P1	P5
P2	P5
P3	P6
P4	P6

COMP	
PX	PY
P1	P2
P1	P3
P2	P3
P2	P4
P3	P5
P4	P5
P5	P6
P1	**P4**
P1	**P5**
P2	**P5**
P3	**P6**
P4	**P6**

Avaliação Semi-Ingênua

NOVA	
PX	PY
P1	P4
P1	P5
P2	P5
P3	P6
P4	P6

COMP	
PX	PY
P1	P2
P1	P3
P2	P3
P2	P4
P3	P5
P4	P5
P5	P6
P1	**P4**
P1	**P5**
P2	**P5**
P3	**P6**
P4	**P6**

Após segunda iteração

NOVA	
PX	PY
P1	P6
P2	P6

COMP	
PX	PY
P1	P2
P1	P3
P2	P3
P2	P4
P3	P5
P4	P5
P5	P6
P1	P4
P1	P5
P2	P5
P3	P6
P4	P6
P1	**P6**
P2	**P6**

Avaliação Semi-Ingênua

- Algoritmo:
  NOVA = EP;
  COMP := NOVA;
  do until NOVA estar vazia;
  NOVA := (NOVA # EP) MINUS COMP;
  COMP := COMP UNION NOVA;
  end;
  DISPLAY := COMP WHERE PX = P#('P1');

- Conclusão
  - Mais eficiente, mas ainda realiza cálculos sobre todos os elementos das relações.

DISPLAY	
PX	PY
P1	P2
P1	P3
P1	P4
P1	P5
P1	P6

Filtragem Estática

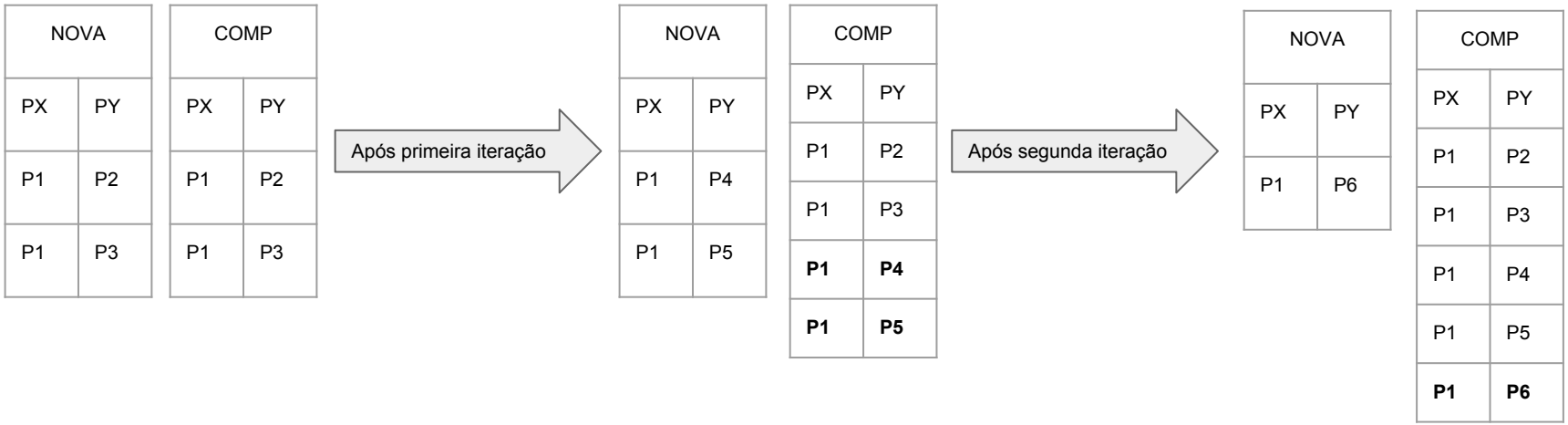
- Algoritmo:

```
NOVA = EP WHERE PX = P#('P1');
COMP := NOVA;
do until NOVA estar vazia;
        NOVA := (NOVA # EP) MINUS COMP;
        COMP := COMP UNION NOVA;
end;
DISPLAY := COMP;
```

- Evita cálculos sobre elementos não relacionados à consulta.

EP	
PX	PY
P1	P2
P1	P3
P2	P3
P2	P4
P3	P5
P4	P5
P5	P6

Filtragem Estática



Filtragem Estática

- Algoritmo:
  NOVA = EP WHERE PX = P#('P1');
  COMP := NOVA;
  do until NOVA estar vazia;
  NOVA := (NOVA # EP) MINUS COMP;
  COMP := COMP UNION NOVA;
  end;
  DISPLAY := COMP;

- Conclusão
  - Refinamento das abordagens anteriores, limitada apenas ao elemento fundamental da consulta

DISPLAY	
PX	PY
P1	P2
P1	P3
P1	P4
P1	P5
P1	P6