Relačné dotazovacie jazyky

- ISBL — Peterlee relational test vehicle (IBM Peterlee UK)
- QUEL INGRES
- QBE (IBM Yorktown Heights)
- SQL (IBM San Jose)

algebra

n-ticový kalkul

kalkul

➔ Aritmetika (nekonečné relácie, vstupná množina)

➔ Priraďovacie a definičné príkazy

➔ Agregačné funkcie

➔ Administrácia a pomocné funkcie

Porovnanie jazykov založených na algebre a jazykov založených na kalkule

- $\{ C: (\exists B) (r(a, B) \wedge s(B, C)) \}$ *1*
- $\{ C: (\exists B) (r(A, B) \wedge s(B, C) \wedge (A = a)) \}$ *2*
- $\Pi_C (\Pi_{AC} ((\sigma_{A=a} r(A, B)) \bowtie s(B, C)))$ *3*
- $\Pi_C ((\sigma_{A=a} r(A, B)) \bowtie s(B, C))$ *4*

Algebra špecifikuje výpočet (je čiastočne operačná)

Kalkul je čiste deklaratívna špecifikácia.

- $q(C) \leftarrow r(a, B), s(B, C)$ (datalog, prolog) *5*

ISBL - information system base language

Relačná algebra	ISBL
$R \cup S$	R + S
R - S	R - S
$R \wedge S$	R . S
$\sigma_F R$	R : F
$\Pi_{A1, .., An} R$	R % A_1, …, A_n
$R \bowtie S$	R * S
Premenovanie:	R % …, A →B, …
Priradenie:	=
Odloženie vyhodnotenia:	N!
Tlač:	List

QUEL - n-ticovo orientovaný kalkul

Schéma príkazov:

range of μ_1 is R_1

.

.

.

range of μ_k is R_k

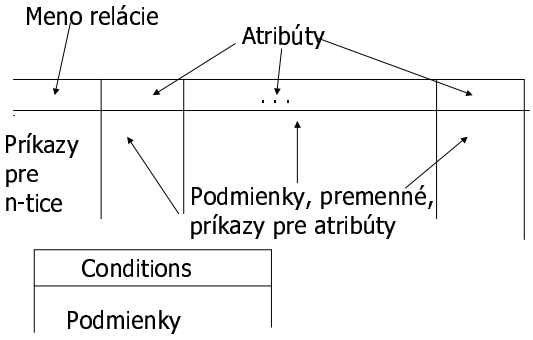
[retrieve | delete | append to]

where $\Psi(\mu_1, …, \mu_k)$

sort

print

QBE - query by example



SQL - structured query language

Schéma dotazu:

Select [**distinct**] [* | zoznam atribútov]

From [zoznam atribútov a ich aliasov]

[**where** podmienky na atribúty]

[**group by** zoznam atribútov

[**having** podmienky na agregácie]

[**order by** zoznam]]

Ďalšiekonštrukcie **SQL** I

- Aritmetika, reťazcové operácie
  +, -, *, /, substring(s, i, j**), like** 'x**%**'
- Agregačné funkcie:
  count, sum, average, min, max
- Dáta definičné príkazy
  Create [table | **view** | **index]** názov
  (deklarácia atribútov)
  Drop [table | **view** | **index]** názov

Ďalšiekonštrukcie **SQL** II

- Dáta manipulačné príkazy

Insert into tabuľka
select

Insert into tabuľka
values

Update tabuľka
set priradenia atribútom
where

Delete from tabuľka
where

Administratívnepríkazy

- **attach** názov databázy
- **set** transaction read only
- **set** line length
- ukončenie transakcie
  commit | **rollback**
- **disconnect**

SQL vložené do hostiteľského jazyka

Príkazy SQL možno vykonávať z hostiteľského jazyka a môžu byť prepojené na hostiteľský jazyk (napr. C, Pascal).

Problém je, že SQL je množinovo orientovaný a programovacie jazyky pracujú s individuálnymi premennými.

Odovzdávanie premenných:

Cursor: to isté ako view, ale dá sa pristupovať k jednotlivým riadkom pomocou operácie FETCH (next | prior).

V programe sú vložene príkazy označené EXEC SQL.
Premenné zabezpečujúce väzbu začínajú dvojbodkou.
Preprocesor preloží vložené príkazy do volania pripravených procedúr.

Príklady príkazow embeded SQL

- Exec SQL execute immediate S
- Exec SQL prepare S from :c
- Exec SQL execute S using :a_1, …, :a_k

} Nie dotaz

```
Exec SQL prepare S from :abraka;
Exec SQL declare C cursor for S;
Exec SQL open C;
Exec SQL whenever not found goto NOMORE;
while (true) {Exec SQL Fetch C into :a1, …, :ak; … };
NOMORE: Exec SQL close C;
```


Príklady: (pôvodné Coddové)

Schéma (databázy), na ktorú sa vzťahujú príklady:

Dodávatelia

Meno	Adresa	Mesto	ČísDod

Súčiastky

ČísSúč	Názov	Farba

Dodáva

ČísDod	ČísSúč

1. Zoznam dodávateľov (meno, mesto), z ktorých každý niečo dodáva.

kalkul

(∃ adresa, ČísDod, ČísSúč)
Dodávatelia(meno, adresa, mesto, ČísDod) ∧
Dodáva(ČísDod, ČísSúč)

algebra

$\Pi_{meno,\ mesto}$ (Dodávatelia ⋈ Dodáva)

SQL

Select distinct meno, mesto
from Dodávatelia, Dodáva
where Dodávatelia.ČísDod = Dodáva.ČísDod

QBE

Dodávatelia	Meno	Adresa	Mesto	ČísDod
	P!		P!	_123

Dodáva	ČísDod	ČísSúč
	_123	

2. Zoznam dodávateľov (meno, mesto), ktorí nič nedodávajú.

kalkul

(∃ adresa, ČísDod) (∀ ČísSúč)
Dodávatelia(meno, adresa, mesto, ČísDod) ∧¬
Dodáva(ČísDod, ČísSúč)

algebra

$\Pi_{meno,\ mesto}$(Dodávatelia ⋈ ($\Pi_{ČísDod}$ Dodávatelia - $\Pi_{ČísDod}$ Dodáva))

SQL

Select distinct meno, mesto
from Dodávatelia
where ČísDod **not in** (**select** ČísDod **from** Dodáva)

QBE

Dodávatelia	Meno	Adresa	Mesto	ČísDod
	P!		P!	_123

Dodáva	ČísDod	ČísSúč
¬	_123	

3. Čísla dodávateľov, ktorí dodávajú súčiastku číslo "15".

kalkul

Dodáva(ČísDod, "15")

algebra

$\Pi_{ČísDod}\ \sigma_{ČísSúč="15"}$ Dodáva

SQL

Select ČísDod
from Dodáva
where ČísSúč = "15"

QBE

Dodáva	ČísDod	ČísSúč
	P!	"15"

4. Čísla dodávateľov, ktorí dodávajú niečo, čo nie je súčiastka číslo "15".

kalkul

(∃x) Dodáva(ČísDod, x) ∧ (x ≠ "15")

algebra

$\Pi_{ČísDod}\ \sigma_{ČísSúč \neq "15"}$ Dodáva

SQL

Select distinct ČísDod
from Dodáva
where ČísSúč ≠ "15"

QBE

Dodáva	ČísDod	ČísSúč
	P!	≠ "15"

5. Čísla dodávateľov, ktorí nedodávajú súčiastku číslo "15".

kalkul

(∃ meno, adresa, mesto)
Dodávatelia(meno, adresa, mesto, ČísDod) ∧
(∀ ČísSúč) Dodáva(ČísDod, ČísSúč) ⇒ ČísSúč ≠ "15"

algebra

$\Pi_{ČísDod}$ Dodávatelia - $\Pi_{ČísDod}\ \sigma_{ČísSúč\ =\ "15"}$ Dodáva

SQL

Select ČísDod **from** Dodávatelia
where ČísDod **not in**
(**select** ČísDod **from** Dodáva **where** ČísSúč = "15")

QBE

Dodávatelia	Meno	Adresa	Mesto	ČísDod
				P!_123

Dodáva	ČísDod	ČísSúč
¬	_123	15

6. Čísla dodávateľov, ktorí dodávajú aj niečo okrem súčiastky číslo "15".

kalkul

(∃x)(Dodáva (ČísDod, "15") ∧
Dodáva(ČísDod, x) ∧ (x ≠ "15"))

algebra

$\Pi_{ČísDod}\ \sigma_{ČísSúč\ =\ "15"}$ Dodáva⋈ $\Pi_{ČísDod}\ \sigma_{ČísSúč \neq "15"}$ Dodáva

SQL

Select distinct ČísDod **from** Dodáva Y
where Dodáva.ČísDod = Y.ČísDod **and**
Dodáva.ČísSúč = "15" **and** Y.ČísSúč ≠ "15"

QBE

Dodáva	ČísDod	ČísSúč
	P!_123	"15"

Dodáva	ČísDod	ČísSúč
	_123	≠ "15"

7. Čísla dodávateľov, ktorí dodávajú len súčiastku číslo "15".

kalkul (∀x) Dodáva(ČísDod, x) ⇒ (x = "15")

algebra $\Pi_{ČísDod}\ \sigma_{ČísSúč\ =\ "15"}$ Dodáva - $\Pi_{ČísDod}\ \sigma_{ČísSúč\neq"15"}$ Dodáva

SQL
Select distinct ČísDod **from** Dodáva
group by ČísDod
having set ČísSúč **in set** ("15")

QBE

Dodáva	ČísDod	ČísSúč
	P!_123	"15"

Dodáva	ČísDod	ČísSúč
¬	_123	≠ "15"

8. Čísla dodávateľov, ktorí dodávajú niečo, ale nedodávajú súčiastku číslo "15".

kalkul (∀x) Dodáva(ČísDod, x) ∧ (x ≠ "15")

algebra $\Pi_{ČísDod}$ Dodáva - $\Pi_{ČísDod}\ \sigma_{ČísSúč\ =\ "15"}$ Dodáva

SQL
Select distinct ČísDod **from** Dodáva
where ČísDod **not in**
(select ČísDod **from** Dodáva **where** ČísSúč = "15" **)**

QBE

Dodáva	ČísDod	ČísSúč
	P!_123	

Dodáva	ČísDod	ČísSúč
¬	_123	"15"

9. Čísla dodávateľov, ktorí dodávajú aspoň súčiastky číslo "12", "13", "15".

kalkul Dodáva(ČísDod, "12") ∧ Dodáva(ČísDod, "13") ∧ Dodáva(ČísDod, "15")

algebra Dodáva : <ČísSúč,{12,13,15}>

SQL
Select ČísDod **from** Dodáva **group by** ČísDod
having set ČísSúč **contains** {12,13,15}

QBE

Dodáva	ČísDod	ČísSúč
	P! _123	"12"
∧	_123	"13"
∧	_123	"15"

10. Čísla dodávateľov, ktorí dodávajú všetky dodávané súčiastky.

kalkul (∀x)(∃y) Dodáva(ČísDod, x) ⇒Dodáva(y,x)

algebra Dodáva(ČísDod, ČísSúč) : $\Pi_{ČísSúč}$ Dodáva

$\Pi_{ČísDod}$ Dodáva -
$\Pi_{ČísDod}$ (($\Pi_{ČísDod}$ Dodáva × $\Pi_{ČísSúč}$ Dodáva) - Dodáva)

SQL
Select ČísDod **from** Dodáva **group by** ČísDod
having set ČísSúč **contains**
(select ČísSúč **from** Dodáva)

11. Zoznam miest odkiaľ prichádza aspoň jedna "červená" súčiastka.

kalkul (∃ meno, adresa, ČísDod, ČísSúč, názov)
Dodávatelia(meno, adresa, mesto, ČísDod) ∧
Dodáva(ČísDod, ČísSúč) ∧
Súčiastky(ČísSúč, názov,"červená")

algebra Π_{mesto} (Dodávatelia⋈Dodáva⋈($\sigma_{farba="červená"}$ Súčiastka))

SQL
Select mesto **from** Dodávatelia, Dodáva, Súčiastka
where farba = "červená"

Delenie v SQL

Select ČísDod **from** Dodáva **group by** ČísDod
having set ČísSúč **contains**
(select ČísSúč **from** Dodáva)

Select ČísDod **from** Dodáva **where** ČísDod **not in**
select Dodáva.ČísDod **from** Dodáva X
where Dodáva. ČísDod, X.ČísSúč **not in**
select * **from** Dodáva

Datalóg – logika a kodátový model

- Základný princíp definuje sa najprv pohľad – program. Potom dotaz – dotaz na čiastočnú zhodu na niektorý predikát definovaného pohľadu.
- Syntax datalógu je podobná syntaxy jazyku logického programovania – prológu.

Príklad:

šéf(Vedúci, Zamestnanec) :– vedie (Vedúci, Zamestnanec).
šéf(Vedúci, Zamestnanec) :–
šéf(Vedúci, Z), vedie(Z, Zamestnanec).
? – šéf(X, peter).

- Premenné
- Databázové – extenzionálne predikáty
- Definované – intencionálne predikáty
- Zabudované predikáty (=, <,>, …)

Terminológia – klauzula a formuly

Fakt = (pozitívna) ***atomická formula***
- $p(a_1, a_2, \ldots, a_n)$
- $p(X_1, X_2, \ldots, X_n)$

Literál = ***atomická formula*** alebo ***negovaná atomická formula*** ($\neg p$, $\overline{p}$)

Klauzula = ***logický súčet*** litearálov (literály pospájané logickou spojkou $\vee$ (***or***)).

Hornova klauzula = klauzula obsahujúca ***najviac jednu*** (nenegovanú) ***atomickú formulu***.

Hornove klauzuly:
- len 1 atomická formula – fakt
- zjednotenie negovaných atomických formúl – podmienka
- práve jedna nenegovaná atomická formula

$p_0 \vee \overline{p_1} \vee \ldots \vee \overline{p_k} \cong p_1 \wedge \ldots \wedge p_k \Rightarrow p_0 \cong p_0 :\!- p_1, \ldots, p_k$

Príklad použitia klauzálnej logiky

Databázová relácia *rodič(x, y)*

súrodenci(x, y) :– rodič(z, x), rodič(z, y), x ≠ y.
bratranci(x, y) :– rodič(u, x), rodič(v, y), súrodenci(u, v).
bratranci(x, y) :– rodič(u, x), rodič(v, y), bratranci(u, v).
príbuzní(x,y) :– súrodenci(x, y).
príbuzní(x,y) :– príbuzní(x, z), rodič(z, y).
príbuzní(x,y) :– príbuzní(z, y), rodič(z, x).

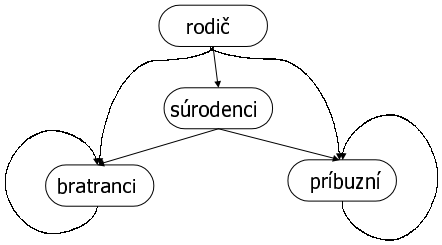
Implementácia aritmetiky zabudovaných funkcií

p(x, z) :– q(x, u, v), z=(u+v)/ 5 . alebo presnejšie

p(x, z) :– q(x, u, v), add(u, v, y), div(y, 5, z).

V zabudovaných funkciách predikátoch treba „strážiť" vstupné množiny.

Graf závislosti predikátov



Uzly sú predikáty. Hrana z uzla p_1 do uzla p_2, ak predikát p_2 je definovaný pomocou predikátu p_1.
Existuje pravidlo s hlavou p_1 ktorého telo obsahuje p_2.

Bezpečné pravidlá – konečné premenné

- Premenná je konečná ak sa vyskytuje v obyčajnom (nenegovanom, nezabudovanom predikáte) tela pravidla alebo v predikáte x = a.
- Ak v predikáte x = y jedna premenná je konečná, potom aj druhá je konečná.

Pravidlo je bezpečné, ak všetky jeho premenné sú konečné.
Dôsledok: Každá premenná v hlave sa musí vyskytovať aj v tele pravidla.

Pravidlo je **rekurzívne**, keď predikát hlavy sa vyskytuje aj v tele pravidla. Program je rekurzívny, keď graf závislosti predikátov obsahuje cyklus.

Výpočet nerekurzívnych programov

Každé pravidlo prerobíme na relačný výraz: náhradou čiarok za prirodzené spojenia.
Problémom je, keď viac pravidiel má ten istý predikát v hlave.
V takomto prípade musíme vytvoriť „rektifikované" pravidlá:
Predikáty v hlave obsahujú tie isté premenné a iba premenné.

Premenovanie, zavedenie premennej

p(x, a) :– … .
p(x, y) :– …, y = a .

Výsledný výraz pre predikát *p* je zjednotenie výrazov, pre telá rektifikovaných pravidiel s hlavou *p*.
Poradie výpočtu predikátov je dané topologickým utriedením grafu závislosti predikátov.
Rektifikáciu a úpravu na algebraické výrazy môžeme urobiť aj pre rekurzívne programy – systém rovníc v relačnej algebre.

Tarského veta o pevnom bode

Úplný zväz: $S = < D, \sqsubseteq, \sqcup, \sqcap, \bot, \top >$

$\sqsubseteq$ čiastočné usporiadanie

$\sqcup$ l.u.b., sup, join $\bot$ dolník

$\sqcap$ g.u.b., inf, meet $\top$ horník

Každá neprázdna množina má l.u.b. (suprémum).

Veta: Nech F je zobrazenie z úplného zväzu S do S také, že $x \sqsubseteq y \Rightarrow F(x) \sqsubseteq F(y)$. Potom F má aspoň jeden pevný bod.

Dôkaz: Nech $U=\{x: x \sqsubseteq F(x)\}$. Množina U je neprázdna, $\bot \in U$.

Označme: $x_0 = \bigsqcup_{x \in U} x$. Pre každé $x \in U$ platí $x \sqsubseteq F(x) \sqsubseteq x_0$.

Preto aj $x_0 \sqsubseteq F(x_0)$ t.j. $x_0 \in U$. Preto aj $F(x_0) \sqsubseteq F(F(x_0))$ a $F(x_0) \in U$. Z toho ale plynie $F(x_0) \sqsubseteq x_0$. Teda $x_0 = F(x_0)$.

Výpočet rekurzívnych programov bez negácie

Systém rovníc: $\vec{P} = \vec{E}(\vec{P}, \vec{R})$, kde $\vec{P}$ sú intencionálne a $\vec{R}$ extenzionálne predikáty.

Riešenie: naivnou iteráciou. Začneme $\vec{P_0} = \vec{0}$. Postupne počítame postupne $P_1, P_2, \ldots$, podľa vzorca:

$$\vec{P}_i = \vec{E}(\vec{P}_{i-1}, \vec{R}),$$

pokiaľ $\vec{P}_i \neq \vec{P}_{i+1}$.

Podľa Tarského vety o pevnom bode tento proces vždy skončí (konverguje). Stačí overiť, že prirodzené spojenie, zjednotenie, projekcia a premenovanie sú „neklesajúce" operácie. Vypočítané riešenie je najmenší pevný bod uvedeného systému rovníc. Z vlastnosti implikácií plynie, že každé riešenie uvedeného systému rovníc musí obsahovať vety vypočítané našim algoritmom.

Inkrementálny výpočet

$P_i = P_{i-1} \cup \Delta_i$

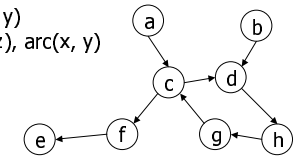
Pri naivnej iterácii sa vyhodnocuje opakovane v každom cykle.

Spojenie:

$(P \cup \Delta) \bowtie (R \cup \Gamma) = P \bowtie R \cup R \bowtie \Delta \cup P \bowtie \Gamma \cup \Delta \bowtie \Gamma$

Seminaivná iterácia: Všetky relácie sa v každom iteračnom kroku rozdelia na nové a staré. Počítajú sa len prirodzené spojenia obsahujúce aspoň jeden prírastkový operand.

Príklad: tc(x, y) :– arc(x, y)
tc(x, y) :– tc(x, z), arc(x, y)



Negácia – stratifikované programy

Rozdiel, negácia nie sú neklesajúce operácie. Vo všeobecnosti programy s negáciou nemusia mať pevný bod.

Stratifikované programy: Ak definícia predikátu p obsahuje negáciu predikátu q, potom v grafe závislosti predikátov neexistuje cesta od p ku q.

Podmienky stratikácie:

- Každému predikátu je priradené celé číslo stratum (vrstva)
- Ak pravidlo p:– … q … . Potom $S(p) \geq S(q)$.
- Ak pravidlo p:– … $\neg q$ … . Potom $S(p) > S(q)$.

Postup stratifikácie: Na začiatku, priradíme všetkým predikátom stratum 1 a spočítame predikáty - n. Prezeráme postupne pravidlá. Ak je porušená niektorá s podmienok stratifikácie, zvýšime stratum predikátu hlavy na najmenšie prirodzené číslo, ktoré ju splňuje. Ak stratum väčšie ako n, program sa nedá stratifikovať. Opakuj zakiaľ sa niečo mení.

Princíp byrokracie predpoklad uzavretého sveta.

Striktne vzaté z Hornových formúl sa nedá odvodiť negácia žiadnej atomickej formuly (negatívny fakt). Aby sme prípadne mohli odvodiť negatívny fakt aplikujeme princíp uzavretého sveta:

Čo sa nedá dokázať z bázy dát neplatí.

Quod non est in datis, non est in mundi.

Pôvodne: actis

Princíp uzavretého sveta vedie k nekonzistencii pri aktualizácii bázy dát. Logika negácii nie je monotónna.

Iné pokusy o negáciu

Negácia pomocou protirečivosti (negation as inconsistency)

$\neg p(x)$ vtedy, keď $db \vee p(x)$ je nekonzistetné.

Toto funguje, ale množina Hornových formúl je vždy neprotirečivá. Potrebujeme nejaké negatívne fakty v báze dát.

p^+, p^- Intencia je p^+ pre pozitívne fakty a p^- pre negatívne fakty. Podmienka $\neg p^+ \vee \neg p^-$.

Takáto negácia je konzistentná, ale dosť slabá. (Matematicky ekvivaletná intuicionistickej logike). Všetko sa dá nasimulovať v datalógu pomocou „rozdvojenia" predikátov.