Úvod do databázových systémov

http://www.dcs.fmph.uniba.sk/~plachetk

/TEACHING/DB1

Tomáš Plachetka

Fakulta matematiky, fyziky a informatiky,
Univerzita Komenského, Bratislava

Zima 2026–2027

Grupovanie a agregácia

Príklad:

vypil(Idn, Alkohol, Mnozstvo) hovorí, koľko ktorého alkoholu sa vypilo počas návštevy Idn v niektorej krčme

navstivil(Idn, Pijan, Krcma) hovorí o návšteve s identifikátorom Idn, ktorého pijana a krčmy sa týka

Dotaz na dvojice **[K, Pocet], ktoré hovoria, koľko rôznych druhov alkoholu sa vypilo v krčme K**

Dotaz na trojice **[K, A, Suma], ktoré hovoria, aké celkové množstvo alkoholu A sa vypilo v krčme K**

Dotaz na dvojice **[K, A], ktoré hovoria, ktoré alkoholy A sa vypili v krčme K v celkovom množstve aspoň 5**

Grupovanie a agregácia: SQL

vypil(Idn, Alkohol, Mnozstvo), **navstivil(Idn, Pijan, Krcma)**

[K, Pocet]: počet rôznych druhov alkoholu vypitých v krčme K

```
SELECT n.Krcma, COUNT(DISTINCT v.Alkohol) as Pocet
FROM navstivil n, vypil v
WHERE n.Idn = v.Idn
GROUP BY n.Krcma
```

V krčme slovak sa vypili 2 rôzne druhy alkoholu, nie 3 (preto ten DISTINCT)

navstivil ⋈ vypil				
Idn	**Alkohol**	**Mnozstvo**	**Pijan**	**Krcma**
1	**pivo**	6	laco	slovak
1	**rum**	1	laco	slovak
2	**rum**	1	laco	slovak

Grupovanie a agregácia: SQL

vypil(Idn, Alkohol, Mnozstvo), **navstivil(Idn, Pijan, Krcma)**

[K, A, Suma]: celkové množstvo alkoholu A vypité v krčme K

```
SELECT n.Krcma, v.Alkohol, SUM(v.Mnozstvo) AS Suma
FROM navstivil n, vypil v
WHERE n.Idn = v.Idn
GROUP BY n.Krcma, v.Alkohol
```

Výsledkom je {[slovak, pivo, 6], [slovak, rum, 2]}

navstivil ⋈ vypil				
Idn	**Alkohol**	**Mnozstvo**	**Pijan**	**Krcma**
1	pivo	**6**	laco	slovak
1	rum	**1**	laco	slovak
2	rum	**1**	laco	slovak

Grupovanie a agregácia: SQL

vypil(Idn, Alkohol, Mnozstvo), navstivil(Idn, Pijan, Krcma)

[K, A]: alkohol A sa v krčme K vypil v celkovom množstve aspoň 5

```
SELECT n.Krcma, v.Alkohol
FROM navstivil n, vypil v
WHERE n.Idn = v.Idn
GROUP BY n.Krcma, v.Alkohol
HAVING SUM(v.Mnozstvo) >= 5
```

navstivil ⋈ vypil				
Idn	**Alkohol**	**Mnozstvo**	**Pijan**	**Krcma**
1	pivo	**6**	laco	slovak
1	rum	**1**	laco	slovak
2	rum	**1**	laco	slovak

Grupovanie a agregácia: relačná algebra

vypil(Idn, Alkohol, Mnozstvo), **navstivil(Idn, Pijan, Krcma)**

[K, Pocet]: počet rôznych druhov alkoholu vypitých v krčme K

$\Gamma_{\text{Krcma, Pocet = COUNT(}\textbf{DISTINCT}\text{ Alkohol)}}$ (navstivil ⋈ vypil)

Alebo, ekvivalentne (všimnite si použitie projekcie π a odstránenia duplikátov δ):

$\Gamma_{\text{Krcma, Pocet = COUNT(Alkohol)}}$ (δ $\pi_{\textbf{Krcma, Alkohol}}$ (navstivil ⋈ vypil))

navstivil ⋈ vypil				
Idn	**Alkohol**	**Mnozstvo**	**Pijan**	**Krcma**
1	**pivo**	6	laco	slovak
1	**rum**	1	laco	slovak
2	**rum**	1	laco	slovak

Grupovanie a agregácia: relačná algebra

vypil(Idn, Alkohol, Mnozstvo), **navstivil(Idn, Pijan, Krcma)**

[K, A, Suma]: celkové množstvo alkoholu A vypité v krčme K

$\Gamma_{\text{Krcma, Alkohol}, \text{Suma} = \text{SUM(Mnozstvo)}}$ (navstivil ⋈ vypil)

Aj toto bude fungovať (keďže v relačnej algebre sa duplikáty neodstraňujú automaticky):

$\Gamma_{\text{Krcma, Alkohol}, \text{Suma} = \text{SUM(Mnozstvo)}}$ $\pi_{\text{Krcma, Alkohol, Mnozstvo}}$ (navstivil ⋈ vypil)

navstivil ⋈ vypil				
Idn	**Alkohol**	**Mnozstvo**	**Pijan**	**Krcma**
1	pivo	**6**	laco	slovak
1	rum	**1**	laco	slovak
2	rum	**1**	laco	slovak

Grupovanie a agregácia: relačná algebra

vypil(Idn, Alkohol, Mnozstvo), **navstivil(Idn, Pijan, Krcma)**

[K, A]: alkohol A sa v krčme K vypil v celkovom množstve aspoň 5

$$\pi_{\text{Krcma, Alkohol}} (\sigma_{\text{Suma >= 5}} (\Gamma_{\text{Krcma, Alkohol, Suma = SUM(Mnozstvo)}} (\text{navstivil} \bowtie \text{vypil})))$$

Výsledkom je {[slovak, pivo]}

navstivil ⋈ vypil				
Idn	**Alkohol**	**Mnozstvo**	**Pijan**	**Krcma**
1	pivo	**6**	laco	slovak
1	rum	**1**	laco	slovak
2	rum	**1**	laco	slovak

Grupovanie a agregácia: relačný kalkul

Grupovanie a agregácia sa chápe ako rozšírenie všeobecného kvantifikátora. Agregačný operátor ♥ viaže (t.j. eliminuje) premenné, ktoré sú vymenované za ♥; a viaže tiež premenné, ktoré sú agregované niektorou agregačnou funkciou. **Ostatné premenné sa použijú na grupovanie.** Výsledkom je predikát s grupovacími premennými a novými premennými, ktoré vznikli agregáciou

(Niektorí autori preferujú symbol ⊎, lenže ten nie je dostupný vo fontoch všetkých súčasných počítačových systémov.)

Relačný kalkul pracuje s **množinami** (bez duplikátov), takže prirodzene nepotrebuje DISTINCT

Grupovanie a agregácia: relačný kalkul

Nech p je predikátová formula s voľnými premennými
G = $\{G_1, \ldots, G_p\}$, **A** = $\{A_1, \ldots, A_q\}$, **B** = $\{B_1, \ldots, B_r\}$,
ktorá zodpovedá relácii P s atribútmi **G** $\cup$ **A** $\cup$ **B**. Množiny **G**, **A**, **B** sú po dvojiciach disjunktné. Nech
$AGG_i \in \{SUM, COUNT, AVG, MIN, MAX\}$, i = 1, ..., q.
Potom

♥ **$B_1, \ldots, B_r, A_1' = AGG_1(A_1), A_q' = AGG_1(A_q)$ (p)**

je predikátová formula s voľnými premennými
$G_1, \ldots, G_p, A_1', \ldots, A_q'$, ktorá zodpovedá relácii

$\Gamma_{G_1, \ldots, G_p, A_1' = AGG_1(A_1), \ldots, A_q' = AGG_q(A_q)}$ **(δ (p))**

Grupovacie premenné sa v relačnom kalkule píšu „naopak“!

Grupovanie a agregácia: relačný kalkul

vypil(Idn, Alkohol, Mnozstvo), **navstivil(Idn, Pijan, Krcma)**

[K, Pocet]: počet rôznych druhov alkoholu vypitých v krčme K

{[K, Pocet]:

♥ Pocet = count(A)

(∃I ∃P ∃M navstivil(I, P, K) ∧ vypil(I, A, M))

}

navstivil ⋈ vypil				
Idn	**Alkohol**	**Mnozstvo**	**Pijan**	**Krcma**
1	**pivo**	6	laco	slovak
1	**rum**	1	laco	slovak
2	**rum**	1	laco	slovak

Grupovanie a agregácia: relačný kalkul

vypil(Idn, Alkohol, Mnozstvo), **navstivil(Idn, Pijan, Krcma)**

[K, A, Suma]: celkové množstvo alkoholu A vypité v krčme K

{[K, A, Suma]:

♥ I, Suma = sum(M)

(∃P navstivil(I, P, K) ∧ vypil(I, A, M))

}

Premennú I je nutné odprojektovať až počas agregácie, nie skôr! (Prečo?)

navstivil ⋈ vypil				
Idn	**Alkohol**	**Mnozstvo**	**Pijan**	**Krcma**
1	pivo	**6**	laco	slovak
1	rum	**1**	laco	slovak
2	rum	**1**	laco	slovak

Grupovanie a agregácia: relačný kalkul

vypil(Idn, Alkohol, Mnozstvo), **navstivil(Idn, Pijan, Krcma)**

[K, A]: alkohol A sa v krčme K vypil v celkovom množstve aspoň 5

{[K, A]:

∃**Suma**

♥ I, Suma = sum(M)

(∃P navstivil(I, P, K) ∧ vypil(I, A, M))

∧ **Suma >= 5**

}

navstivil ⋈ vypil				
Idn	**Alkohol**	**Mnozstvo**	**Pijan**	**Krcma**
1	pivo	**6**	laco	slovak
1	rum	**1**	laco	slovak
2	rum	**1**	laco	slovak

Grupovanie a agregácia: Datalog

Špeciálny predikát subtotal, ktorého **prvým argumentom je n-árny predikát p** s navzájom disjunktnými množinami argumentov G_1, ..., G_p, A_1, ..., A_q, B_1, ..., B_r. Premenné B_1, ..., B_r mimo subtotalu neexistujú

Druhým argumentom subtotal je zoznam [G_1, ..., G_p].

Tretím argumentom subtotal je zoznam

[A1' = $agg_1(A_1)$, ..., A_q' = $agg_q(A_q)$], kde agg_i je niektorá z agregačných funkcií.

Výsledkom je predikát, ktorý zodpovedá formuli

♥ **B_1, ..., B_r, A_1' = $AGG_1(A_1)$, A_q' = $AGG_1(A_q)$ p(...)**

Relačný kalkul pracuje s **množinami** (bez duplikátov), takže prirodzene nepotrebuje DISTINCT

Grupovanie a agregácia: Datalog

vypil(Idn, Alkohol, Mnozstvo), **navstivil(Idn, Pijan, Krcma)**

[K, Pocet]: počet rôznych druhov alkoholu vypitých v krčme K

vypite_alkoholy(K, A) ← navstivil(I, _, K), vypil(I, A, _).

answer(K, Pocet) ←

subtotal(vypite_alkoholy(K, A), [K], [Pocet = count(A)]).

navstivil ⋈ vypil				
Idn	**Alkohol**	**Mnozstvo**	**Pijan**	**Krcma**
1	**pivo**	6	laco	slovak
1	**rum**	1	laco	slovak
2	**rum**	1	laco	slovak

Grupovanie a agregácia: Datalog

vypil(Idn, Alkohol, Mnozstvo), **navstivil(Idn, Pijan, Krcma)**

[K, A, Suma]: celkové množstvo alkoholu A vypité v krčme K

ucet(I, K, A, M) ← navstivil(I, _, K), vypil(I, A, M).

answer(K, A, Suma) ←

subtotal(ucet(_, K, A, M), [K, A], [Suma = sum(M)]).

Premennú I je nutné odprojektovať až počas agregácie, nie skôr!

navstivil ⋈ vypil				
Idn	**Alkohol**	**Mnozstvo**	**Pijan**	**Krcma**
1	pivo	**6**	laco	slovak
1	rum	**1**	laco	slovak
2	rum	**1**	laco	slovak

Grupovanie a agregácia: Datalog

vypil(Idn, Alkohol, Mnozstvo), **navstivil(Idn, Pijan, Krcma)**

[K, A]: alkohol A sa v krčme K vypil v celkovom množstve aspoň 5

ucet(I, K, A, M) ← navstivil(I, _, K), vypil(I, A, M).

answer(K, A) ←

subtotal(ucet(_, K, A, M), [K, A], [Suma = sum(M)]),

Suma >= 5.

navstivil ⋈ vypil				
Idn	**Alkohol**	**Mnozstvo**	**Pijan**	**Krcma**
1	pivo	**6**	laco	slovak
1	rum	**1**	laco	slovak
2	rum	**1**	laco	slovak

Rekapitulácia dotazovacích jazykov

Dotazy sa dajú vyjadriť v 4 jazykoch, ktoré majú pre bezpečné dotazy rovnakú vyjadrovaciu silu (až na spracovanie duplikátov a podobne), t.j. je možné napísať kompilátor z ľubovoľného jazyka do ľubovoľného

- **Relačný kalkul**: bezpečné predikátové formuly
- **Datalog**: bezpečné predikátové formuly v "kanonickej" forme
- „Kanonický" **SQL**:

  SELECT... FROM... WHERE (+NOT EXISTS)...

  GROUP BY... HAVING (+NOT EXISTS)...

  UNION...
- **Relačná algebra**: výpočty nad reláciami, s operátormi zjednotenia, rozdielu, joinu, (kartézskeho súčinu), selekcie, projekcie, premenovania a agregácie

Takmer... Zatiaľ **nevieme vyjadriť rekurziu v relačnej algebre, t.j. nevieme počítať rekurzívne dotazy**

Datalog→relačná algebra

Uvažujme nasledujúci Datalogový (bezpečný) program nad databázou
navstivil(Idn, Pijan, Krcma), vypil(Idn, Alkohol, Mnozstvo)

answer(P, K) ← navstivil(I, P, K), vypil(I, rum, _), not pilvodku(P).
answer(P, K) ← navstivil(I, P, K), vypil(I, pivo, _), not pilvodku(P).
pilvodku(P) ← navstivil(I, P, _), vypil(I, vodka, _).

Predikát answer(P, K) je splnený pre dvojice [P, K] také, že pijan P pri niektorej návšteve krčmy K vypil rum alebo pivo, a zároveň P nikdy (pri žiadnej návšteve ktorejkoľvek krčmy) nevypil vodku.

Graf závislostí IDB predikátov (predikátov definovaných v programe):

answer ⇄ **pilvodku**

Datalog→relačná algebra

lubi(Pijan, Alkohol), capuje(Krcma, Alkohol, Cena), navstivil(Idn, Pijan, Krcma), vypil(Idn, Alkohol, Mnozstvo).

answer(P, K) ← navstivil(I, P, K), vypil(I, rum, _), not pilvodku(P).

answer(P, K) ← navstivil(I, P, K), vypil(I, pivo, _), not pilvodku(P).

pilvodku(P) ← navstivil(I, P, _), vypil(I, vodka, _).

$$\textbf{answer} \overset{\neg}{\leftarrow} \textbf{pilvodku}$$

Preklad začína s pilvodku, lebo pilvodku nezávisí od iných IDB predikátov, t.j. v grafe závislostí predikátov do vrcholu pilvodku nevedie žiadna hrana. Všetky pozitívne relačné podciele sa spoja joinom, väzby argumentov sa vyjadria v podmienke joinu. Napokon sa výsledok projekciou prispôsobí hlave pravidla:

pilvodku := $\pi_{\text{n.Pijan}}(\rho_{\text{n}}(\textbf{navstivil}) \bowtie_{\text{n.Idn=v.Idn} \wedge \text{ v.Alkohol='vodka'}} \rho_{\text{v}}(\textbf{vypil}))$

Datalog→relačná algebra

lubi(Pijan, Alkohol), capuje(Krcma, Alkohol, Cena), navstivil(Idn, Pijan, Krcma), vypil(Idn, Alkohol, Mnozstvo).

answer(P, K) ← navstivil(I, P, K), vypil(I, rum, _), not pilvodku(P).

answer(P, K) ← navstivil(I, P, K), vypil(I, pivo, _), not pilvodku(P).

pilvodku(P) ← navstivil(I, P, _), vypil(I, vodka, _).

Predikát answer je definovaný viacerými pravidlami. Po unifikácii hláv pravidiel (ak hlavy nie sú rovnaké, vieme ich urobiť rovnaké premenovaním premenných) sa každé pravidlo preloží zvlášť a výsledky sa zjednotia do relácie answer:

answer := (*preklad 1. pravidla*) ∪ (*preklad 2. pravidla*)

Datalog→relačná algebra

lubi(Pijan, Alkohol), capuje(Krcma, Alkohol, Cena), navstivil(Idn, Pijan, Krcma), vypil(Idn, Alkohol, Mnozstvo).

answer(P, K) ← navstivil(I, P, K), vypil(I, rum, _), not pilvodku(P).

answer(P, K) ← navstivil(I, P, K), vypil(I, pivo, _), not pilvodku(P).

pilvodku(P) ← navstivil(I, P, _), vypil(I, vodka, _).

Pozitívna časť 1. pravidla sa preloží ako join, negácia ako množinový rozdiel. Aby sa rozdiel dal urobiť, treba z toho joinu projekciou nechať len atribút Pijan:

$\pi_{n.Pijan}\,(\rho_n(navstivil) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='rum'} \rho_v(vypil)) - pilvodku$

Lenže vo výsledku chceme dvojice [Pijan, Krcma], a atribút Krcma sme tou projekciou stratili. Vieme ho však dostať späť opätovným (prirodzeným) prijoinovaním pozitívnej časti pravidla:

$\pi_{n.Pijan,\ n.Krcma}$ (

$(\pi_{n.Pijan}\,(\rho_n(navstivil) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='rum'} \rho_v(vypil)) - pilvodku) \bowtie$

$(\rho_n(navstivil) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='rum'} \rho_v(vypil))))$

Datalog→relačná algebra

lubi(Pijan, Alkohol), capuje(Krcma, Alkohol, Cena), navstivil(Idn, Pijan, Krcma), vypil(Idn, Alkohol, Mnozstvo).

answer(P, K) ← navstivil(I, P, K), vypil(I, rum, _), not pilvodku(P).

answer(P, K) ← navstivil(I, P, K), vypil(I, pivo, _), not pilvodku(P).

pilvodku(P) ← navstivil(I, P, _), vypil(I, vodka, _).

Plán výpočtu programu v relačnej algebre (počíta správny výsledok, hoci možno nie efektívne):

pilvodku := $\pi_{n.Pijan}(\rho_n(navstivil) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='vodka'} \rho_v(vypil))$

answer := $\pi_{n.Pijan, n.Krcma}$ (

($\pi_{n.Pijan}(\rho_n(navstivil) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='rum'} \rho_v(vypil)) - pilvodku) \bowtie$

($\rho_n(navstivil) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='rum'} \rho_v(vypil)))$

$\cup$ $\pi_{n.Pijan, n.Krcma}$ (

($\pi_{n.Pijan}(\rho_n(navstivil) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='pivo'} \rho_v(vypil)) - pilvodku) \bowtie$

$\rho_n(navstivil)))$

Datalog→relačná algebra: antijoin

lubi(Pijan, Alkohol), capuje(Krcma, Alkohol, Cena), navstivil(Idn, Pijan, Krcma), vypil(Idn, Alkohol, Mnozstvo).

answer(P, K) ← navstivil(I, P, K), vypil(I, rum, _), not pilvodku(P).

answer(P, K) ← navstivil(I, P, K), vypil(I, pivo, _), not pilvodku(P).

pilvodku(P) ← navstivil(I, P, _), vypil(I, vodka, _).

Trik s prijoinovaním atribútov "stratených" pri výpočte rozdielu relácií je častý. Operátor rozdielu množín (−) sa dá nahradiť všeobecnejším operátorom ▷, ktorému sa hovorí **antijoin** (vo výsledku antijoinu sú všetky záznamy z ľavej relácie, ktoré sa nejoinujú so žiadnym záznamom pravej relácie):

pilvodku := $\pi_{n.Pijan}(\rho_n(\text{navstivil}) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='vodka'} \rho_v(\text{vypil}))$

answer := $\pi_{n.Pijan,\ n.Krcma}$ (

$(\rho_n(\text{navstivil}) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='rum'} \rho_v(\text{vypil})) \triangleright \text{pilvodku}))$

$\cup\ \pi_{n.Pijan,\ n.Krcma}$ (

$(\rho_n(\text{navstivil}) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='pivo'} \rho_v(\text{vypil})) \triangleright \text{pilvodku}))$

Datalog→relačná algebra: resume

Ak je graf závislostí predikátov acyklický, tak vieme nájsť také usporiadanie, že každý predikát závisí len od EDB a od predošlých predikátov. V tomto poradí sa budú počítať relácie prislúchajúce tým predikátom v relačnej algebre

- Ak je viacej pravidiel, ktoré definujú jeden predikát, vypočítaj každé zvlášť a výsledky spoj zjednotením $\cup$
- Pozitívnu časť tela pravidla prelož ako join $\bowtie$, negatívnu ako antijoin $\triangleright$. Výsledok prispôsob hlave pravidla vhodnou projekciou

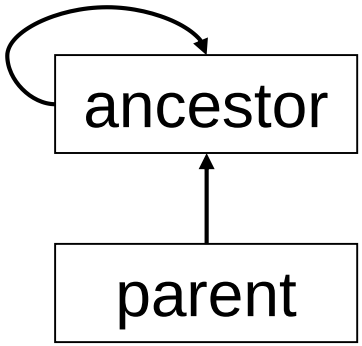
„Malichernosti“ s premenovaním relácií, konštantami, väzbami medzi atribútmi atď. sa vyriešia podobne ako pri preklade Datalog → SQL

Rekurzia: Datalog

Príklad (predkovia):

ancestor(X, A) :- parent(X, A).

ancestor(X, A) :- parent(X, P), ancestor(P, A).



Rekurzívne programy obsahujú cyklus v grafe závislostí predikátov (hrana z uzla p do uzla q, ak existuje pravidlo s hlavou q, ktorého telo obsahuje p)

Komplikovanejšia rekurzia (vyjadruje to isté čo predošlý program):

ancestor(X, A) :- parent(X, A).

ancestor(X, A) :- ancestor(X, Y), ancestor(Y, A).

Rekurzia: relačná algebra

Operátor **fixpoint** (tranzitívny uzáver): φ

- φ(<postupnosť priradení>) iteruje priradenia (každé raz), kým sa relácie v tej postupnosti priradení menia (t.j. iterácia pokračuje ak sa zmení obsah aspoň 1 relácie)

Príklad:

ancestor := parent(X, A)

ancestor := ancestor $\cup$ $\pi_{X, A}$(ancestor $\bowtie$ parent)

ancestor := ancestor $\cup$ $\pi_{X, A}$(ancestor $\bowtie$ parent)

...

sa zapíše ako

ancestor := parent

φ(ancestor := ancestor $\cup$ $\pi_{X, A}$(ancestor $\bowtie$ parent))

Alebo, ekvivalentne (takýto preklad pravidiel funguje všeobecne)

ancestor := {}

φ(ancestor := parent $\cup$ $\pi_{X, A}$(ancestor $\bowtie$ parent))

Rekurzia: relačný kalkul

V relačnom kalkule netreba zavádzať špeciálny operátor na vyjadrenie rekurzie. Rekurzia sa vyjadrí prirodzeným spôsobom

Príklad (predkovia Johna):

ancestor(X, A) :- parent(X, A).

ancestor(X, A) :- parent(X, P), ancestor(P, A).

answer(A) :- ancestor(john, A).

$\{A: ancestor(john, A) \wedge$

$(\forall X \forall A\ (parent(X, A) \Rightarrow ancestor(X, A))) \wedge$

$(\forall X \forall A \forall P\ ((parent(X, A) \wedge ancestor(P, A)) \Rightarrow ancestor(X, A))))\}$

Rekurzia: SQL

V SQL-99 bola pridaná rekurzia: rekurzívne VIEW, resp. rekurzívne WITH.

Bohužiaľ, niektoré systémy neimplementujú WITH RECURSIVE. Spoliehajú na cykly resp. rekurziu v externých jazykoch mimo SQL (napr. PHP). Existujúce systémy, ktoré WITH RECURSIVE implementujú (Oracle, Firebird, DB2), neimplementujú plnú sémantiku (implementujú len rekurziu cez 1 reláciu). Do PostgreSQL bol WITH RECURSIVE dodaný v 2011.

Príklad:

```
WITH RECURSIVE ancestor AS (
  SELECT p.X FROM parent p
  UNION
  SELECT p.X FROM parent p, ancestor a WHERE p.Y=a.X
)
SELECT a.X FROM ancestor a
```

Výpočet bezpečných Datalogových dotazov

Každému (IDB) predikátu p_i v programe priraď reláciu P_i

Algoritmus výpočtu výsledku dotazu (naivná evaluácia):

1. Pre všetky p_i polož $P_i := \emptyset$
2. do
   {
   aplikuj výpočet pravidiel každého predikátu p_i práve raz, **v ľubovoľnom poradí** (t.j. pre každý predikát vypočítaj zjednotenie joinov/antijoinov s použitím aktuálneho obsahu relácií v tele pravidiel), výsledok priraď relácii P_i
   } while (niektorá relácia P_i sa zmenila)
3. Aplikuj výslednú selekciu a projekciu danú dotazom

V kroku 2 sa počíta "výsledok celého programu" a **až v kroku 3 sa tento výsledok uložený v reláciách P_i zredukuje na výsledok daného dotazu** (ktorý sa týka len jednej z týchto relácií). Toto môže byť neefektívne, ale funguje korektne pre ľubovoľný program

Výpočet bezpečných Datalogových dotazov

Problém: krok 2 v naivnej evaluácii nemusí vždy skončiť

Príklad:

man(barber). man(mayor).

shaves(barber, X) ← man(X), not shaves(X, X).

Počítajme naivnou evaluáciou (relácia man je EDB, tá sa nemení). Predikátu shaves(., .) priradíme reláciu Shaves(X, Y). Iterujeme pravidlo pre Shaves (jediné)

Shaves := Man $\triangleright_{\text{Man.X = Shaves.X} \wedge \text{Man.X = Shaves.Y}}$ **Shaves**

Shaves := ∅

Shaves := {[barber, barber], [barber, mayor]}

Shaves := {[barber, mayor]}

Shaves := {[barber, barber], [barber, mayor]}

Shaves := {[barber, mayor]}

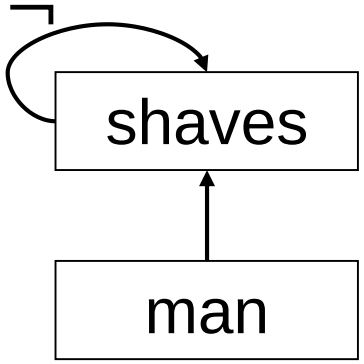
…

Výpočet bezpečných Datalogových dotazov

Problém: krok 2 v naivnej evaluácii nemusí vždy skončiť

?- shaves(barber, mayor) neskončí, ak sa počíta naivnou evaluáciou

Graf závislostí predikátov:



Pre istú podtriedu dotazov je garantované, že výpočet naivnou evaluáciou skončí

Výpočet pevného bodu: Tarskeho veta

Tarskeho veta o pevnom bode pre úplné zväzy: ak *f* je neklesajúce zobrazenie nad úplným zväzom (t.j. $x \leq y \Rightarrow f(x) \leq f(y)$), tak potom *f* má aspoň jeden pevný bod p, pre ktorý platí f(p) = p

Z Tarskeho vety o pevnom bode vyplýva, že **iteratívny výpočet konverguje k pevnému bodu pre "obvyklé" (aj rekurzívne) dotazy**, keďže kartézsky súčin, join, zjednotenie, projekcia a premenovanie sú „neklesajúce" operácie a extenzionálne relácie sú konečné. **K „neklesajúcim" operáciám patrí aj *stratifikovaná negácia***

Problém je s nestratifikovanou negáciou (ba aj s bežnou aritmetikou)

Stratifikovaná negácia

Definícia: program je **stratifikovaný**, keď existuje priradenie S(.), ktoré každému predikátu priradí celé číslo, tzv. stratum (vrstva) tak, že platí:

- Ak p je definovaný pomocou q, tak $S(p) \geq S(q)$
- Ak p je definovaný pomocou $\neg$q, tak $S(p) > S(q)$

Platí: ak je program stratifikovaný a obsahuje n predikátov, tak tým predikátom možno priradiť strata od 1 po n

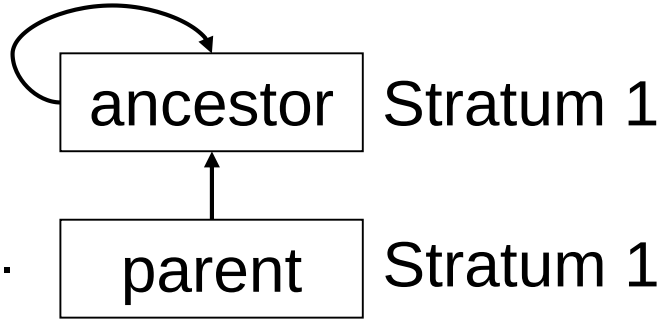
Algoritmus stratifikácie:

1. Všetkým predikátom priraď stratum 1
2. Ak niektoré pravidlo porušuje podmienku stratifikácie, tak zvýš stratum predikátu v hlave toho pravidla na najmenšie celé číslo, ktoré tú podmienku spĺňa
3. Ak je niektoré stratum väčšie ako počet predikátov, tak program sa nedá stratifikovať

Stratifikovaná negácia

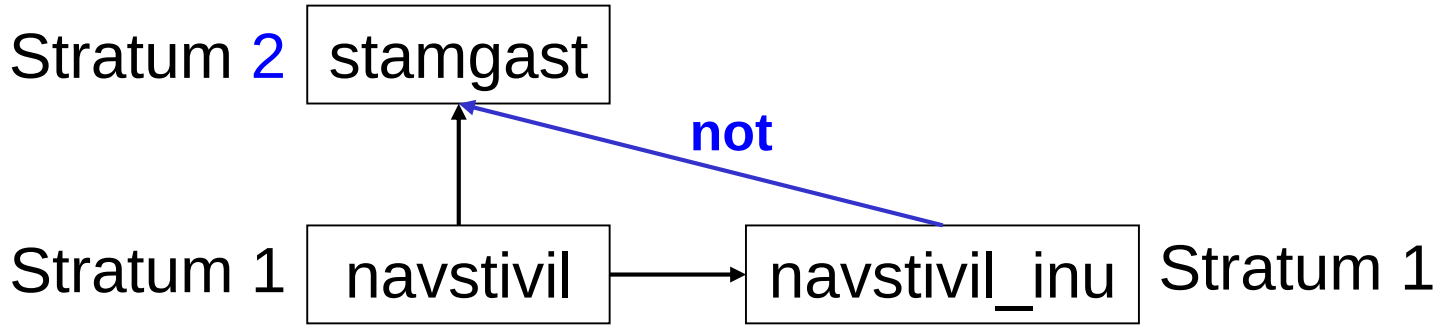
Príklady:

ancestor(X, A) :- parent(X, A).
ancestor(X, A) :- parent(X, P), ancestor(P, A).



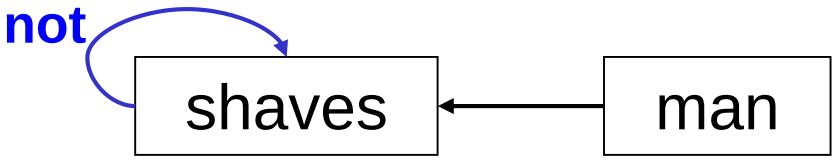
---

stamgast(P) :- navstivil(_, P, K), **not** navstivil_inu(P, K).
navstivil_inu(P, K) :- navstivil(_, P, K), navstivil(_, P, K2), K ≠ K2.



---

shaves(barber, X) ← man(X), **not** shaves(X, X).



Nedá sa stratifikovať

Výpočet stratifikovaných Datalogových dotazov

Každému (IDB) predikátu s_i v programe priraď reláciu S_i

Metóda výpočtu výsledku dotazu (naivná evaluácia):
Začni s prázdnymi reláciami S_i
Vypočítaj relácie S_i priradené predikátom v stratum 1
Vypočítaj relácie S_i priradené predikátom v stratum 2
Vypočítaj relácie S_i priradené predikátom v stratum 3
...

Poradie výpočtu S_i v rámci jedného strata:

- Pri výpočte každého strata preferuj najskôr predikáty, ktoré nezávisia od iných IDB predikátov. Relácie pre tieto predikáty stačí vypočítať len raz
- V prípade rekurzívnej definície iteruj výpočet (súčasne) cez všetky predikáty v cykle grafu závislostí, až kým sa prislúchajúce relácie S_i prestanú meniť. Tieto relácie sa potom už nikdy nebudú meniť

Výpočet pevného bodu: príklad (vrstovníci)

Relácia p(Child, Parent)

Napr. p={[**C**laudia, **A**dam], [**D**ieter, **A**dam], [**D**ieter, **B**ritta], [**E**rik, **B**ritta], [**F**ridda, **C**laudia], [**G**isela, **C**laudia], [**H**einz, **D**ieter], [**I**ngo, **D**ieter], [**I**ngo, **E**rik], [**F**ridda, **E**rik], [**J**ürgen, **F**ridda], [**J**ürgen, **H**einz], [**K**laus, **G**isela], [**K**laus, **I**ngo]}

Dvojice ľudí X a Y, ktorí sú v rovnakej generácii (**s**ame **g**eneration): buď X a Y majú spolu dieťa, alebo ich rodičia sú v rovnakej generácii:

sg(X, Y) :- p(C, X), p(C, Y).
sg(X, Y) :- p(X, PX), p(Y, PY), sg(PX, PY).

Treba nájsť všetky dvojice X a Y, pre ktoré platí sg(X,Y)

Výpočet pevného bodu: iterácia (naivná evaluácia)

sg(X, Y) :- p(C, X), p(C, Y).
sg(X, Y) :- p(X, PX), p(Y, PY), sg(PX, PY).

Relačná algebra, fixpoint operátor (premenovania sú vynechané):
sg(X, Y) := {};
φ (sg(X, Y) := δ ($\pi_{X,Y}$(p(C, X) $\bowtie$ p(C, Y)) $\cup$
$\pi_{X,Y}$(p(X, PX) $\bowtie$ sg(PX, PY) $\bowtie$ p(Y, PY))));

Relačná algebra, naivná evaluácia:
sg(X, Y) := {};
do /* Tento cyklus reprezentuje operátor φ */
{
old_sg(X, Y) := sg(X, Y);
sg(X, Y) := δ ($\pi_{X,Y}$(p(C, X) $\bowtie$ p(C, Y)) $\cup$
$\pi_{X,Y}$(p(X, PX) $\bowtie$ sg(PX, PY) $\bowtie$ p(Y, PY)));
}
while old_sg(X, Y) <> sg(X, Y);

Výpočet pevného bodu: iterácia (naivná evaluácia)

```
sg(X, Y) :- p(C, X), p(C, Y).
sg(X, Y) :- p(X, PX), p(Y, PY), sg(PX, PY).
```

SQL:

```
WITH RECURSIVE sg AS (
SELECT p1.Parent AS X, p2.Parent AS Y
FROM p p1, p p2
WHERE p1.Child = p2.Child
UNION
SELECT p1.Child AS X, P2.Child AS Y
FROM p p1, p p2, sg s
WHERE p1.Parent = s.X AND p2.Parent = s.Y)

SELECT DISTINCT s.X, s.Y
FROM sg s
```

Výpočet pevného bodu: iterácia (naivná evaluácia)

```
sg(X, Y) :- p(C, X), p(C, Y).
sg(X, Y) :- p(X, PX), p(Y, PY), sg(PX, PY).
```

SQL, naivná evaluácia s použitím cyklu mimo SQL:

```
sg := {};
do /* Tento cyklus je implementovaný v operátore φ */
{
        old_sg := sg;
        sg :=
        (SELECT p1.Parent AS X, p2.Parent AS Y
        FROM p p1, p p2
        WHERE p1.Child = p2.Child
        UNION
        SELECT p1.Child AS X, P2.Child AS Y
        FROM p p1, p p2, sg s
        WHERE p1.Parent = s.X AND p2.Parent = s.Y);
}
while (old_sg <> sg);
```

Výpočet pevného bodu: iterácia (naivná evaluácia)

sg(X, Y) :- p(C, X), p(C, Y).
sg(X, Y) :- p(X, PX), p(Y, PY), sg(PX, PY).

p = {[**C**laudia, **A**dam], [**D**ieter, **A**dam], [**D**ieter, **B**ritta], [**E**rik, **B**ritta], [**F**ridda, **C**laudia], [**G**isela, **C**laudia], [**H**einz, **D**ieter], [**I**ngo, **D**ieter], [**I**ngo, **E**rik], [**F**ridda, **E**rik], [**J**ürgen, **F**ridda], [**J**ürgen, **H**einz], [**K**laus, **G**isela], [**K**laus, **I**ngo]}

i	**|sg|**	**sg(X, Y)**
1	19	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II
2	29	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II, CD, DC, FG, FI, GF, HI, IF, IH, JJ, KK
3	33	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II, CD, DC, FG, FI, GF, HI, IF, IH, JJ, KK, GH, HG, JK, KJ
4	33	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II, CD, DC, FG, FI, GF, GH, HG, HI, IF, IH, JJ, JK, KJ, KK

Výpočet pevného bodu: iterácia (seminaivná evaluácia)

Optimalizácia: nepočítať opakovane všetky dvojice v každom kroku. Stačí počítať (a pridávať) **len nové dvojice**

Naivná evaluácia: 19+29+33+33=**114 vypočítaných dvojíc**

Seminaivná evaluácia: 19+10+4+0=**33 vypočítaných dvojíc**

i	**|sg|**	**sg(X, Y)**	**|Δsg|**	**Δsg(X, Y)**
1	19	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II	19	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II
2	29	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II, CD, DC, FG, FI, GF, HI, IF, IH, JJ, KK	10	CD, DC, FG, FI, GF, HI, IF, IH, JJ, KK
3	33	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II, CD, DC, FG, FI, GF, HI, IF, IH, JJ, KK, GH, HG, JK, KJ	4	GH, HG, JK, KJ
4	33	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II, CD, DC, FG, FI, GF, HI, IF, IH, JJ, KK, GH, HG, JK, KJ	0	

Výpočet pevného bodu: iterácia (seminaivná evaluácia)

sg(X, Y) :- p(C, X), p(C, Y).
sg(X, Y) :- p(X, PX), p(Y, PY), sg(PX, PY).

Relačná algebra, seminaivná evaluácia (diferenčná schéma):

$sg(X, Y) := \emptyset;$
$\Delta sg(X, Y) := \emptyset;$
do /* Tento cyklus reprezentuje operátor φ */
{
$\Delta sg(X, Y) := (\pi_{X,Y}(p(C, X) \bowtie p(C, Y)) \cup$
$\pi_{X,Y}(p(X, PX) \bowtie p(Y, PY) \bowtie \Delta sg(PX, PY)))$
$- sg(X, Y);$
$sg(X, Y) := sg(X, Y) \cup \Delta sg(X, Y);$
}
while $(\Delta sg(X, Y) <> \emptyset)$;

Výpočet pevného bodu: iterácia (seminaivná evaluácia)

sg(X, Y) :- p(C, X), p(C, Y).
sg(X, Y) :- p(X, PX), p(Y, PY), sg(PX, PY).

Seminaivná evaluácia, optimalizovaná verzia (prvé pravidlo prispeje do Δsg iba raz):

sg(X, Y) := $\emptyset$;
Δsg(X, Y) := $\pi_{X,Y}$(p(C, X) $\bowtie$ p(C, Y));
do /* Tento cyklus reprezentuje operátor Φ */
{
    Δsg(X, Y) := ($\pi_{X,Y}$(p(X, PX) $\bowtie$ p(Y, PY) $\bowtie$ Δsg(PX, PY)))
        – sg(X, Y);
    sg(X, Y) := sg(X, Y) $\cup$ Δsg(X, Y);
}
while (Δsg(X, Y) <> $\emptyset$);

Výpočet pevného bodu: iterácia (seminaivná evaluácia)

Embedded SQL, seminaivná evaluácia (s iteráciou mimo SQL):

```
sg := ∅;
Δsg := ∅;
do /* Tento cyklus reprezentuje operátor φ */
        Δsg :=
        SELECT newsg.X, newsg.Y
        FROM
                (SELECT p1.Parent AS X, p2.Parent AS Y
                FROM p p1, p p2 J
                WHERE p1.Child = p2.Child
                UNION
                SELECT p1.Child AS X, P2.Child AS Y
                FROM p p1, p p2, sg s
                WHERE p1.Parent = s.X AND p2.Parent = s.Y) newsg
        WHERE [s.X, s.Y] NOT IN
                (SELECT oldsg.X, oldsg.Y FROM sg oldsg);
        sg(X, Y) := sg(X, Y) ∪ Δsg(X, Y);
while (Δsg(X, Y) <> ∅);
```

Ešte príklad: emp(Id, Name, Mgr), Datalog

emp(Employee_Id, Employee_Name, Manager_Id)
Nájdite všetkých nadriadených zamestnanca BLAKE (priamych aj nepriamych).

Datalog:
answer(N) ← emp(Id, N, _), super_blake(Id).
super_blake(Id) ← emp(_, blake, Id).
super_blake(Id) ← emp(Mid, _, Id), super_blake(Mid).

Lenže obyčajne sa snažíme programovať všeobecné predikáty (super_blake je príliš jednoúčelový). Nasledujúca verzia je všeobecnejšia:
super(Id, Mid) ← emp(Id, _, Mid).
super(Id, Mid) ← emp(Id, _, Mid2), super(Mid2, Mid).
answer(N) ← emp(Mid, Sname, _), emp(Id, blake, _), super(Id, Mid).

Pred výpočtom dotazu „? answer(N)“ dobrý kompilátor/optimalizátor **automaticky** transformuje tento všeobecný program na program, ktorý je veľmi podobný tomu prvému programu

Ešte príklad: emp(Id, Name, Mgr), relačná algebra

Nájdite všetkých nadriadených zamestnanca BLAKE (priamych aj nepriamych).

super(Id, Mid) ← emp(Id, _, Mid).
super(Id, Mid) ← emp(Id, _, Mid2), super(Mid2, Mid).
answer(Mname) ← emp(Mid, Mname, _), emp(Id, blake, _), super(Id, Mid).

Relačná algebra:

φ (super := $\pi_{Id,\ Mgr}$ emp $\cup$ $\pi_{emp.Id,\ super.Mgr}$ emp $\bowtie_{emp.Mgr\ =\ super.Id}$ super;
answer := $\sigma_{Name='blake'}$ (ρ_{e2} (emp)) $\bowtie_{e2.Id\ =\ s.Id}$ ρ_{s} (super) $\bowtie_{e1.Id\ =\ s.Mgr}$ ρ_{e1} (emp))
Finálny dotaz:
answer

Ešte príklad: emp(Id, Name, Mgr), relačný kalkul

Nájdite všetkých nadriadených zamestnanca BLAKE (priamych aj nepriamych).

super(Id, Mid) $\leftarrow$ emp(Id, _, Mid).
super(Id, Mid) $\leftarrow$ emp(Id, _, Mid2), super(Mid2, Mid).
answer(Mname) $\leftarrow$ emp(Mid, Mname, _), emp(Id, blake, _), super(Id, Mid).

Relačný kalkul:
{Mname:
($\exists$Mid$\exists$X1$\exists$X2$\exists$Id emp(Mid, Mname, X1) $\wedge$ emp(Id, blake, X2) $\wedge$ super(Id, Mid))
$\wedge$
($\forall$Id$\forall$Mid$\forall$N (emp(Id, N, Mid) $\Rightarrow$ super(Id, Mid))) $\wedge$
($\forall$Id$\forall$Mid$\forall$Mid2$\forall$N ((emp(Id, N, Mid2) $\wedge$ super(Mid2, Mid)) $\Rightarrow$ super(Id, Mid))))}

Ešte príklad: emp(Id, Name, Mgr), WITH RECURSIVE

Nájdite všetkých nadriadených zamestnanca BLAKE (priamych aj nepriamych).

super(Id, Mid) ← emp(Id, _, Mid).
super(Id, Mid) ← emp(Id, _, Mid2), super(Mid2, Mid).
answer(Mname) ← emp(Mid, Mname, _), emp(Id, blake, _), super(Id, Mid).

SQL s WITH RECURSIVE:

```
with recursive super as
(select e.Id, e.Mgr
from emp e)
union
(select e.Id, s.Mgr
from emp e, super s
where e.Mgr = s.Id),

select e1.Name
from emp e1, emp e2, super s
where e1.Id = s.Mgr and e2.Id = s.Id and e2.Name = ‘blake’
```

Ešte príklad: emp(Id, Name, Mgr), SQL bez WITH

Nájdite všetkých nadriadených zamestnanca BLAKE (priamych aj nepriamych).

super(Id, Mid) ← emp(Id, _, Mid).
super(Id, Mid) ← emp(Id, _, Mid2), super(Mid2, Mid).
answer(Mname) ← emp(Mid, Mname, _), emp(Id, blake, _), super(Id, Mid).

SQL bez WITH:

```
create temporary table super1 as
select e.Id, e.Mgr
from emp e;

create temporary table super2 as
((select e.Id, e.Mgr
from emp e)
union
(select e.Id, s.Mgr
from emp e, super1 s
where e.Mgr = s.Id));
```

Ešte príklad: emp(Id, Name, Mgr), SQL bez WITH, pokr.

Nájdite všetkých nadriadených zamestnanca BLAKE (priamych aj nepriamych).

super(Id, Mid) ← emp(Id, _, Mid).
super(Id, Mid) ← emp(Id, _, Mid2), super(Mid2, Mid).
answer(Mname) ← emp(Mid, Mname, _), emp(Id, blake, _), super(Id, Mid).

SQL bez WITH, pokračovanie:

...

```
create temporary table super10 as
((select e.Id, e.Mgr
from emp e)
union
(select e.Id, s.Mgr
from emp e, super9 s
where e.Mgr = s.Id));
```

Ešte príklad: emp(Id, Name, Mgr), SQL bez WITH, pokr.

Nájdite všetkých nadriadených zamestnanca BLAKE (priamych aj nepriamych).

super(Id, Mid) ← emp(Id, _, Mid).
super(Id, Mid) ← emp(Id, _, Mid2), super(Mid2, Mid).
answer(Mname) ← emp(Mid, Mname, _), emp(Id, blake, _), super(Id, Mid).

SQL bez WITH, finálny SELECT (pri hĺbke rekurzie 10):

```
select e1.Name
from emp e1, emp e2, super10 s
where e1.Id = s.Mgr and e2.Name = 'blake' and e2.Id = s.Id;
```

Pre **ľubovoľnú fixnú** hĺbku rekurzie je možné rekurziu simulovať aj bez použitia WITH RECURSIVE (musí byť známe horné ohraničenie tej hĺbky)

Vyskúšajte na databáze emp (stačí hĺbka rekurzie 3, netreba 10)

Ešte príklad: emp(Id, Name, Mgr), SQL bez WITH, ∆-opt

Nájdite všetkých nadriadených zamestnanca BLAKE (priamych aj nepriamych).

super(Id, Mid) ← emp(Id, _, Mid).
super(Id, Mid) ← emp(Id, _, Mid2), super(Mid2, Mid).
answer(Mname) ← emp(Mid, Mname, _), emp(Id, blake, _), super(Id, Mid).

SQL bez WITH, seminaivná evaluácia (efektívnejšia ako predošlá naivná):

```
create temporary table super2 as
((select e.Id, e.Mgr
from emp e)
union
(select e.Id, s.Mgr
from emp e, super1 s
where e.Mgr = s.Id and not exists (
  select *
  from super1 s1
  where s1.Id = e.Id and s1.Mgr = s.Mgr)));
```

… podobne pre super3 až super10