Úvod do databázových systémov

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

/TEACHING/DB1

Tomáš Plachetka

Fakulta matematiky, fyziky a informatiky,
Univerzita Komenského, Bratislava

Zima 2021–2022

Relácie

Definícia: **relácia** je podmnožina kartézskeho súčinu množín.
Formálne, n-árna relácia r je definovaná ako
$\langle r \subseteq D_1 \times D_2 \times \ldots \times D_n, D_1, D_2, \ldots, D_n \rangle$

Prvkami relácie sú usporiadané n-tice (hovorí sa im tiež n-tuples, tuples, záznamy). Jednotlivým zložkám tých n-tíc sa hovorí atribúty (argumenty) relácie. Množinám $D_1, D_2, \ldots, D_n$ sa hovorí domény (typy) atribútov relácie r

Kartézsky súčin nie je komutatívny, takže **na poradí atribútov záleží**. V relačnom kalkule sa atribúty adresujú pozíciou

Ale ak atribúty pomenujeme, tak ich môžeme ľubovoľne premiešať. Tak sa to robí v SQL, atribúty sa adresujú menom.
Relácie r(Mesto, Psc) a r(Psc, Mesto) sú vtedy ekvivalentné

Relácie

Ako vieme popísať **konkrétnu** reláciu? Ak je konečná, tak môžeme **vymenovať** všetky n-tice, ktoré do nej patria. Napríklad boolovskú reláciu f vieme popísať takto:
f = {[0, 0, 0], [0, 1, 0], [1, 0, 0], [1, 1, 1]}.
Pre úplnosť treba dodať, že $f \subseteq \{0, 1\} \times \{0, 1\} \times \{0, 1\}$, t.j., že atribúty X, Y a Z môžu mať v relácii f(X, Y, Z) len hodnoty 0 a 1

Domény atribútov obvykle nebudeme uvádzať. Niekedy sú však dôležité. Napríklad,
{[X, Y] $\in$ N x N: X < Y}
je **iná** relácia ako
{[X, Y] $\in$ R x R: X < Y}

Predikáty

Definícia: Nech r je n-árna relácia
$\langle r \subseteq D_1 \times D_2 \times \ldots \times D_n, D_1, D_2, \ldots, D_n \rangle$.
Potom **n-árny predikát prislúchajúci relácii** r je funkcia
$r_p: D_1 \times D_2 \times \ldots \times D_n \rightarrow$ boolean,
pričom $r_p(A_1, ..., A_n)$ = TRUE práve vtedy keď $[A_1, ..., A_n] \in r$

Nebudeme rozlišovať medzi menom relácie a menom predikátu, ktorý tej relácii prislúcha
Napríklad, reláciu r(Mesto, Psc) „stotožníme" s predikátom r(Mesto, Psc) tak, že relácia r(Mesto, Psc) bude obsahovať práve tie n-tice [M, P], pre ktoré platí predikát r(M, P)

Konvenciou bude, že mená predikátov a relácií budú písané malými písmenami. Konštanty v argumentoch predikátov budú písané malými písmenami, premenné veľkými písmenami

Relačný kalkul: dotazy

Dotazy v relačnom kalkule sú formuly logiky 1.rádu. Databáza (extenzionálna) je konečná množina predikátov, ktoré sú splnené pre **konečné množiny** n-tíc

Výsledok dotazu je usporiadaná množina hodnôt voľných premenných, po ktorých dosadení tá formula platí. Ak dotaz nemá voľné premenné, tak výsledkom dotazu je TRUE resp. FALSE

Príklady dotazov a ich výsledkov nad databázou
f(., ., .) = {[0, 0, 0], [0, 1, 0], [1, 0, 0], [1, 1, 1]}:

f(0, 0, 0) = TRUE	$\exists X\ f(X, X, X) = TRUE$
f(0, 0, 1) = FALSE	$\{X: \exists Y\ f(X, X, Y) \wedge f(Y, Y, X)\} = \{0, 1\}$
f(0, 0, 2) = FALSE	
{[X, Y]: f(X, Y, 1)} = {[1, 1]}	{X: f(X, Y, 4)} je nezmysel
{[X, Y]: f(X, Y, 2)} = ⊘	{[X, Y, Z]: ¬ f(X, Y, Z)} = ???
{X: f(0, X, X)} = {0}	$\forall X\ \forall Y\ \forall Z\ f(X, Y, Z) =$???

Relačný kalkul: bezpečnosť dotazov (safety)

Obmedzenia, ktoré umožňujú výpočet dotazov:
V dotazoch dovolíme len doménovo nezávislé formuly
(ich výsledok závisí nie od domény, ale **len** od obsahu databázy)

Neexistuje algoritmus, ktorý pre ľubovoľnú formulu rozhodne či je doménovo nezávislá (hoci pre niektoré konkrétne formuly to rozhodnúť vieme). Nepomáha ani to, že formuly vieme ekvivalentne transformovať do dohodnutej normálnej formy

Ale pre istú podtriedu doménovo nezávislých formúl sa to dá:

Definícia: Premenná V_i formuly ϕ je ohraničená (range-restricted), ak vieme algoritmicky vypočítať konečnú množinu hodnôt R_i takú, že po dosadení ľubovoľnej hodnoty $v \notin R_i$ za V_i formula ϕ nie je splniteľná

Definícia: Formula je bezpečná (safe), ak každá jej premenná je ohraničená

Platí: **Každá bezpečná formula je doménovo nezávislá**

Relačný kalkul: bezpečnosť dotazov (safety)

Ak sa dohodneme na vhodnej normálnej forme bezpečných formúl, tak pre ľubovoľnú formulu vieme algoritmicky (syntaktickou analýzou) rozhodnúť, či je v danej normálnej forme

Výsledok ľubovoľnej bezpečnej formuly vieme vypočítať
Napríklad tak, že vyskúšame všetky možné dosadenia do všetkých premenných (premenných aj "rozumných dosadení" je konečne veľa)

Ku každej doménovo nezávislej formuli existuje (a dá sa skonštruovať) ekvivalentná bezpečná formula

Relačný kalkul: definície predikátov

V relačnom kalkule vieme z existujúcich predikátov definovať (konštruovať) nové predikáty **implikáciou**

Príklad: daná je konečná relácia krat(., ., .), napr. malá násobilka
{Z: ∃X ∃Y krat(X, Y, Z) ∧ X≠1 ∧ Y≠1} je množina zložených čísiel
Takto sa definuje jej charakteristický predikát zlozene_cislo(Z):
∀Z (∃X ∃Y krat(X, Y, Z) ∧ X≠1 ∧ Y≠1**) ⇒ zlozene_cislo(Z)**

Definujme predikát prvocislo(.):
∀Z ((∃X ∃Y krat(X, Y, Z)) ∧ ¬ zlozene_cislo(Z)) **⇒ prvocislo(Z))**
Toto je trochu nepresné, treba ešte pridať definíciu zlozene_cislo:
(∀Z (∃X ∃Y krat(X, Y, Z) ∧ X≠1 ∧ Y≠1) ⇒ zlozene_cislo(Z)) ∧
(∀Z ((∃X ∃Y krat(X, Y, Z)) ∧ ¬ zlozene_cislo(Z)) ⇒ prvocislo(Z)))

Dotaz na prvočísla (skrátený zápis, ktorý je presný v konjunkcii s predošlou formulou):
{Z: prvocislo(Z)}

Relačný kalkul: rekurzívne definície predikátov

Príklad: daný je predikát r(Rodic, Dieta)
Chceme definovať predikát p(Predok, Potomok), ktorý je pravdivý pre také dvojice, že osoba Predok je predkom osoby Potomok.

Sú 2 prípady:
1. Osoba Predok je rodičom osoby Potomok (priamy potomok)
2. Osoba Predok je predkom *nejakej* osoby Rodic, pričom Potomok je dieťaťom osoby Rodic

$\forall$Predok $\forall$Potomok (
(
r(Predok, Potomok) /* prípad 1 */
$\vee$ ($\exists$ Rodic p(Predok, Rodic) $\wedge$ r(Rodic, Potomok)) /* prípad 2 */
)
$\Rightarrow$ p(Predok, Potomok))

Jazyk relačného kalkulu netreba kvôli rekurzii rozširovať

Datalog

Datalog je programovací jazyk syntaxou podobný jazyku Prolog (deklaratívne resp. logické programovanie)

Program je konečná množina pravidiel. Syntax pravidiel:
<atóm>.
<hlava> ← <telo>.

<hlava>: <atóm>
<telo>: <atóm> | not <atóm> | <telo>, <atóm> | <telo>, not <atóm>

- Žiadne domény (typy)
- **V hlave pravidla smie byť iba jeden pozitívny atóm**
- **V tele** pravidla sú pozitívne alebo negované **atomické podciele** oddelené čiarkami (',' znamená '∧', '←' znamená implikáciu)
- Časť riadku začínajúca znakom ‘%’ označuje komentár

Datalog: fakty (extenzionálna databáza, EDB)

Príklad (malá násobilka):

krat(1, 1, 1). krat(1, 2, 2). krat(1, 3, 3). krat(1, 4, 4). krat(1, 5, 5).
krat(1, 6, 6). krat(1, 7, 7). krat(1, 8, 8). krat(1, 9, 9). krat(1, 10, 10).
krat(2, 1, 2). krat(2, 2, 4). krat(2, 3, 6). krat(2, 4, 8). krat(2, 5, 10).
krat(2, 6, 12). krat(2, 7, 14). krat(2, 8, 16). krat(2, 9, 18). krat(2, 10, 20).
krat(3, 1, 3). krat(3, 2, 6). krat(3, 3, 9). krat(3, 4, 12). krat(3, 5, 15).
krat(3, 6, 18). krat(3, 7, 21). krat(3, 8, 24). krat(3, 9, 27). krat(3, 10, 30).
krat(4, 1, 4). krat(4, 2, 8). krat(4, 3, 12). krat(4, 4, 16). krat(4, 5, 20).
krat(4, 6, 24). krat(4, 7, 28). krat(4, 8, 32). krat(4, 9, 36). krat(4, 10, 40).
krat(5, 1, 5). krat(5, 2, 10). krat(5, 3, 15). krat(5, 4, 20). krat(5, 5, 25).
krat(5, 6, 30). krat(5, 7, 35). krat(5, 8, 40). krat(5, 9, 45). krat(5, 10, 50).
krat(6, 1, 6). krat(6, 2, 12). krat(6, 3, 18). krat(6, 4, 24). krat(6, 5, 30).
krat(6, 6, 36). krat(6, 7, 42). krat(6, 8, 48). krat(6, 9, 54). krat(6, 10, 60).
krat(7, 1, 7). krat(7, 2, 14). krat(7, 3, 21). krat(7, 4, 28). krat(7, 5, 35).
krat(7, 6, 42). krat(7, 7, 49). krat(7, 8, 56). krat(7, 9, 63). krat(7, 10, 70).
krat(8, 1, 8). krat(8, 2, 16). krat(8, 3, 24). krat(8, 4, 32). krat(8, 5, 40).
krat(8, 6, 48). krat(8, 7, 56). krat(8, 8, 64). krat(8, 9, 72). krat(8, 10, 80).
krat(9, 1, 9). krat(9, 2, 18). krat(9, 3, 27). krat(9, 4, 36). krat(9, 5, 45).
krat(9, 6, 54). krat(9, 7, 63). krat(9, 8, 72). krat(9, 9, 81). krat(9, 10, 90).
krat(10, 1, 10). krat(10, 2, 20). krat(10, 3, 30). krat(10, 4, 40). krat(10, 5, 50).
krat(10, 6, 60). krat(10, 7, 70). krat(10, 8, 80). krat(10, 9, 90). krat(10, 10, 100).

Datalog: pravidlá (intenzionálna databáza)

Príklad (malá násobilka):

zlozene_cislo(Z) ←
krat(X, Y, Z),
not X = 1,
not Y = 1.

prvocislo(Z) ←
krat(X, Y, Z),
not zlozene_cislo(Z).

Prepis do ASCII (‘←’ sa píše ako ‘:-’, ‘not’ ako ‘\+’, singleton premenné ako ‘_’):

```
zlozene_cislo(Z) :-
        krat(X, Y, Z),
        \+ X = 1,
        \+ Y = 1.

prvocislo(Z) :-
        krat(_, _, Z),
        \+ zlozene_cislo(Z).
```

Datalog: pravidlá (intenzionálna databáza)

Program v Datalogu sa skladá z **pozitívnych** faktov a z pravidiel s **pozitívnou hlavou** (negatívne veci sa nedajú definovať a nedá sa na ne pýtať)
Program sa spúšťa tým, že mu kladieme dotazy

?- zlozene_cislo(Z).
[4], [6], [8], [9], [10], [12], [14], [15], [16], [18], [20], [21], [24], [25], [27], [28], [30], [32], [35], [36], [40], [42], [45], [48], [49], [50], [54], [56], [60], [63], [64], [70], [72], [80], [81], [90], [100]
37 tuples.

?- zlozene_cislo(33).
No

?- prvocislo(33).
No

zlozene_cislo(Z) ←
krat(X, Y, Z),
not X = 1,
not Y = 1.

prvocislo(Z) ←
krat(X, Y, Z),
not zlozene_cislo(Z).

1. Prečo 33 nie je ani zložené číslo, ani prvočíslo?
2. Zmeňme to druhé pravidlo na **prvocislo(Z) ← not zlozene_cislo(Z).**
Bude potom platiť prvocislo(33)?

Bezpečnosť pravidiel a programov v Datalogu (safety)

Definícia: **Program je bezpečný**, ak každé jeho pravidlo je bezpečné

Definícia: **Pravidlo je bezpečné**, ak každá premenná použitá kdekoľvek v tom pravidle je použitá aj v nejakom **pozitívnom relačnom** podcieli (relačný podcieľ je predikát definovaný pravidlom programu)

Bezpečnosť Datalogového pravidla, resp. programu sa dá strojovo overiť

Výsledky dotazov na bezpečné programy vieme strojovo počítať

Príklad (Ullman/Widom):
p(X, Y) ← q(X, Z), ¬ r(W, X, Z), W < Y.
Toto pravidlo nie je bezpečné (safe) z niekoľkých dôvodov:
• Premenná Y je použitá v hlave pravidla, ale v žiadnom relačnom podcieli. Toto vo všeobecnosti znemožňuje vymenovať konečnú množinu hodnôt pre Y, ktoré sú „kandidátmi“ pre odvodenie p(X, Y)
• Premenná Y je použitá v aritmetickom podcieli (<), ale v žiadnom relačnom podcieli
• Premenná W je použitá len v negovanom relačnom podcieli a v aritmetickom podcieli

Datalog: sémantika (o čom to je)

Datalogový program je konjunkciou implikácií (pravidlo v Datalogu je implikácia), pričom všetky premenné pravidla sú kvantifikované všeobecným kvantifikátorom

Inak povedané, každé pravidlo v Datalogu je implikácia:
- v predpoklade (na pravej strane pravidla) je konjunkcia pozitívnych a negovaných atómov (predikátov s argumentami)
- v dôsledku (na ľavej strane pravidla) je jeden pozitívny atomický predikát

Pravidlá bez pravej strany sú len skratkami za implikácie:
krat(1, 1, 1).
sa dá ekvivalentne zapísať ako
krat(1, 1, 1) ← true.
alebo
krat(X, Y, Z) ← X = 1, Y = 1, Z = 1.

V relačnom kalkule:
$\forall X\ \forall Y\ \forall Z\ ((X = 1 \wedge Y = 1 \wedge Z = 1) \Rightarrow krat(X, Y, Z))$.

Datalog: sémantika (o čom to je)

Prečo stačí na definíciu implikácia?

Príklad: Uvažujme pre jednoduchosť program s jediným pravidlom
(*true* je predikát bez argumentov, vždy pravdivý):
krat(1, 1, 1) ← true.

Intuitívne chceme vyjadriť, že krat(., ., .) platí **len** pre [1, 1, 1]

Formula true ⇒ krat(1, 1, 1) nevylučuje, že platí napr. krat(jumbo, $\Re$, π)
Na druhej strane, nevylučuje ani to, že krat(jumbo, $\Re$, π) neplatí

Čo vlastne platí? Čo je výsledkom dotazu „?- krat(X, Y, Z)“ pre tento program?

Datalog: sémantika (o čom to je)

Žiadne obavy. Pre program ***true ⇒ krat(1, 1, 1)*** je výsledkom dotazu ***?- krat(X, Y, Z)*** trojica **[1, 1, 1] a nič iné**

V Datalogu platí len to, čo „platiť musí“ (dá sa odvodiť, t.j. dokázať z programu)

krat(jumbo, ℜ, π) sa z tohto programu **odvodiť nedá** (rovnako dobre môže a nemusí platiť). Tým pádom to **neplatí**

To zodpovedá intuícii. „Nezmysly“ sa nedostanú do výsledku dotazov na bezpečné programy

Definície s ⇒ sa v „bežných prípadoch" správajú presne tak ako definície s ⇔

V okrajových (aj prakticky relevantných) prípadoch sa ⇒ správa dokonca v istom zmysle lepšie

Datalog: sémantika (o čom to je), Russelov “paradox”

“Holič holí všetkých mužov v meste, ktorí neholia sami seba“

man(barber). % Konštanty pomenované rôznymi menami sú rôzne
man(mayor). % T.j. barber ≠ mayor
shaves(barber, X) ← man(X), ¬ shaves(X, X).

Tento (rekurzívny) program je bezpečný, lebo X sa v poslednom pravidle vyskytuje v pozitívnom relačnom podcieli man(X)

Holí holič starostu?
?- shaves(barber, mayor).
Po dosadení do pravidla:
shaves(barber, mayor) ← man(mayor), ¬ shaves(mayor, mayor).
Výsledkom tohto dotazu je true, lebo shaves(mayor, mayor) sa zo žiadneho pravidla nedá odvodiť

Kto holí starostu?
?- shaves(X, mayor). % {X: shaves(X, mayor)}
Výsledkom je {barber}

Datalog: sémantika (o čom to je), Russelov “paradox”

man(barber).
man(mayor).
shaves(barber, X) ← man(X), ¬ shaves(X, X).

?- shaves(barber, barber).
Po dosadení do posledného pravidla:
shaves(barber, barber) ← man(barber), ¬ **shaves(barber, barber)**.

Sú len 2 možnosti:
• shaves(barber, barber) = false
Lenže potom ¬ shaves(barber, barber) = true, čo porušuje implikáciu v poslednom pravidle
• **shaves(barber, barber) = true**
Potom ¬ shaves(barber, barber) = false, čo neodporuje žiadnemu pravidlu

Lenže **shaves(barber, barber) nie je možné dokázať** (hoci je true)
Preto prirodzená odpoveď na tento dotaz je "neviem" ("neviem" nie je true)

Čo znamená "dokázať"? Čo je "dôkaz"?

Datalog a relačný kalkul

	Relačný kalkul	**Datalog**
Konštanta	a, b, jozo, ..., 1, 2, 3, ...	a, b, jozo, ..., 1, 2, 3, ...
Premenná	X, Y, Z, ...	X, Y, Z, ..., _, ...
Zložený term	$f(t_1, t_2, ..., t_n)$	$f(t_1, t_2, ..., t_n)$
Atomická formula	$p(t_1, t_2, ..., t_n)$	$p(t_1, t_2, ..., t_n)$
Konjunkcia	$p \wedge q$	p, q
Disjunkcia	$p \vee q$	dve pravidlá
Implikácia	$p \Rightarrow q$	$q \leftarrow p$ (resp. q :- p)
Negácia	$\neg p$	not p (resp. \+ p)
Existenčný kvantifikátor	$\exists X\ p(X)$	
Všeobecný kvantifikátor	$\forall X\ p(X)$	
Zátvorky	(...)	

Zdá sa, že v Datalogu niečo chýba. V skutočnosti **nič nechýba**

Datalog a relačný kalkul: ∀, ∃, ∨

Príklad: lubi(Pijan, Alkohol), navstivil(Pijan, Krcma)
"Nájdite (všetkých) **pijanov**."

Dohodnime sa, že pijan je ten, kto ľúbi aspoň jeden alkohol **alebo** navštívil aspoň jednu krčmu
Chceme definovať predikát pijan(.), ktorý je true (len) pre pijanov. To „alebo“ sa vyjadrí dvomi pravidlami

pijan(P) ← lubi(P, _). % Pijan je ten, kto ľúbi nejaký alkohol
pijan(P) ← navstivil(P, _). % Pijan je ten, kto navštívil nejakú krčmu

V relačnom kalkule:
$[\forall P \ \forall A \ (lubi(P, A) \Rightarrow pijan(P))] \wedge [\forall P \ \forall K \ (navstivil(P, K) \Rightarrow pijan(P))]$
$\equiv$
$\forall P \ [(\exists A \ lubi(P, A) \vee \exists K \ navstivil(P, K)) \Rightarrow pijan(P)]$

"Strojové chápanie" kvantifikátorov v Datalogu je také, ako v tej prvej formuli:
Každá premenná pravidla je kvantifikovaná univerzálne (∀)
Formuly pre jednotlivé pravidlá sú spojené konjunkciou (∧)

Datalog a relačný kalkul: ∀, ∃, ∨

Príklad: lubi(Pijan, Alkohol), navstivil(Pijan, Krcma)
"Nájdite (všetkých) pijanov."
Pijan je ten, kto ľúbi aspoň jeden alkohol alebo navštívil aspoň jednu krčmu

pijan(P) ← lubi(P, A).
pijan(P) ← navstivil(P, K).

Napriek "strojovému chápaniu", pri tvorbe Datalogových pravidiel je metodicky (ľudsky) lepšie uvažovať o existencii či neexistencii niečoho. Takto:
pijan(P) platí, ak existuje alkohol A taký, že platí lubi(P, A)
a tiež
pijan(P) platí, ak existuje krčma K taká, že platí navstivil(P, K)

Tento spôsob uvažovania snáď vystihuje najlepšie táto formula (ktorá je tým predošlým ekvivalentná)
∀P [((∃A lubi(P, A)) ⇒ pijan(P)) ∧ ((∃K navstivil(P, K)) ⇒ pijan(P))]

Datalog a relačný kalkul: ∀, ∃, ∨

Príklad: lubi(Pijan, Alkohol), navstivil(Pijan, Krcma)
"Nájdite (všetkých) pijanov."
Pijan je ten, kto ľúbi aspoň jeden alkohol alebo navštívil aspoň jednu krčmu

Ak v relačnom kalkule nechceme vytvárať predikát pijan(.), ale priamo vyjadriť množinu pijanov, tak sa tá formula zjednoduší:
{P: ∃A lubi(P, A) ∨ ∃K navstivil(P, K)}

S definíciou predikátu pijan(.) bude ten dotaz vyzerať takto:
{P: pijan(P) ∧ ∀P2 [(∃A lubi(P2, A) ∨ ∃K navstivil(P2, K)) ⇒ pijan(P2)]

V Datalogu pre zlepšenie čitateľnosti môžeme premenné, ktoré sa v pravidle vyskytujú práve raz, nahradiť anonymnými '_' (na menách týchto premenných nezáleží, hovorí sa im "anonymné premenné" alebo "singleton premenné"):

```
pijan(P) ←
        lubi(P, _).
pijan(P) ←
        navstivil(P, _).

?- pijan(P).
```

Datalog a relačný kalkul: ¬, (,)

Príklad: lubi(Pijan, Alkohol), navstivil(Pijan, Krcma)
"Nájdite množinu pijanov, ktorí navštívili **každú** krčmu."

Metodicky (ľudsky) vhodné preformulovať dotaz, ktorý obsahuje slová ako napr. "každý", "všetci", "žiaden", ... Keď uvažujeme o existencii či neexistencii niečoho, tak tieto slová **skrývajú v sebe negáciu**

"Nájdite množinu pijanov P takú, že **neplatí**, že pijan P **nenavštívil niektorú** krčmu."
Ešte lepšie je:
"Nájdite množinu pijanov P takú, že **neplatí**, že **existuje** krčma K taká, že pijan P **nenavštívil** K." (Musíme sa ešte dohodnúť na definícii krčmy. Povedzme, že krčma je to, čo navštívil aspoň jeden pijan. Hociktorý pijan, nie nutne pijan P.)

V relačnom kalkule:

$$\{P: (\exists A\ lubi(P, A) \vee \exists K\ navstivil(P, K)) \wedge \quad /* \text{ safety: P je pijan } */$$
$$\neg$$
$$(\exists K\ \exists P2\ navstivil(P2, K) \wedge \quad /* \text{ safety: K je krčma } */$$
$$\neg\ navstivil(P, K)$$
$$)$$
$$\}$$

Datalog a relačný kalkul: ¬, (,)

Príklad: lubi(Pijan, Alkohol), navstivil(Pijan, Krcma)
"Nájdite množinu pijanov, ktorí navštívili **každú** krčmu."

"Nájdite množinu pijanov P takú, že **neplatí** (pre pijana P), že existuje krčma K taká, že pijan P **nenavštívil** K."

V Datalogu nie sú zátvorky. Musíme definovať **pomocné predikáty** pre "neplatí, že ...":

```
pijan(P) ← lubi(P, _).
pijan(P) ← navstivil(P, _).

navstivil_kazdu(P) ←
        pijan(P),           /* safety */
        not nenavstivil_niektoru(P).

nenavstivil_niektoru(P) ←
        pijan(P),           /* safety */
        navstivil(_, K),    /* safety */
        not navstivil(P, K).

?- navstivil_kazdu(P).
```