7/1/2015 Úvod do databáz, skúškový test, max 25 bodov, 90 min

1. Daná je databáza (bez duplikátov a null hodnôt): capuje(Krcma, Alkohol),
lubi(Pijan, Alkohol), navstivil(Idn, Pijan, Krcma), vypil(Idn, Alkohol, Mnozstvo).
Platí: Idn → Pijan, Krcma; Idn, Alkohol → Mnozstvo; Mnozstvo > 0.
a) Nájdite trojice [P, A1, A2] také, že pijan P ľúbi oba alkoholy A1, A2; a zároveň pri každej návšteve krčmy vypil práve jeden z týchto alkoholov. (Trojice s A1=A2 nemajú byť vo výsledku.) Sformulujte tento dotaz v Datalogu **(2)** a v relačnom kalkule **(2)**.

Datalog:

```
answer(P, A1, A2) ←
        lubi(P, A1),
        lubi(P, A2),
        not A1 = A2,
        not vypil_zle(P, A1, A2).

vypil_zle(P, A1, A2) ←
        navstivil(I, P, _),
        vypil(I, A1, _),
        vypil(I, A2, _).

vypil_zle(P, A1, A2) ←
        navstivil(I, P, _),
        lubi(P, A1),            /* safety */
        lubi(P, A2),            /* safety */
        not v(I, A1),
        not v(I, A2).

v(I, A) ←
        vypil(I, A, _).
```

Podmienka A1 ≠ A2 v prvom pravidle ošetruje prípad, keď pijan P nenavštívil žiadnu krčmu. Ak pijan P nenavštívil žiadnu krčmu, chceme aby answer(P, A1, A2) platil len pre dvojice rôznych alkoholov A1 a A2, ktoré ten pijan ľúbi.

Relačný kalkul:

$$
\begin{aligned}
&\{[P, A1, A2]: \\
&\quad lubi(P, A1) \wedge lubi(P, A2) \wedge (A1 \neq A2) \wedge \\
&\quad \neg \text{ /* } vypil\_zle(P, A1, A2) \text{ */} \\
&\quad (\\
&\qquad (\\
&\qquad\quad \exists I \, \exists K \, \exists M1 \, \exists M2 \\
&\qquad\quad navstivil(I, P, K) \wedge vypil(I, A1, M1) \wedge vypil(I, A2, M2) \\
&\qquad) \\
&\qquad \vee \\
&\qquad (\\
&\qquad\quad \exists I \, \exists K \\
&\qquad\quad navstivil(I, P, K) \wedge \\
&\qquad\quad \neg \text{ /* } v(I, A1) \text{ */} \\
&\qquad\quad (\\
&\qquad\qquad \exists M \\
&\qquad\qquad vypil(I, A1, M) \\
&\qquad\quad) \\
&\qquad\quad \wedge \\
&\qquad\quad \neg \text{ /* } v(I, A2) \text{ */} \\
&\qquad\quad (\\
&\qquad\qquad \exists M \\
&\qquad\qquad vypil(I, A2, M) \\
&\qquad\quad) \\
&\qquad) \\
&\quad) \\
&\}
\end{aligned}
$$

b) Nájdite dvojice [K, A] pre ktoré platí: krčmu K, ktorá čapuje alkohol A, navštívilo aspoň 200 (rôznych) pijanov; a zároveň sa alkohol A v krčme K vypil v celkovom množstve najviac 10. Sformulujte tento dotaz v Datalogu **(2)** a SQL **(2)**.

Datalog:

```
answer(K, A) ←
        aspon200p(K),
        najviac10v(K, A).

aspon200p(K) ←
        subtotal(n(K, P), [K], [C = count(P)]),
        C >= 200.

n(K, P) ←
        navstivil(_, P, K).

najviac10v(K, A) ←
        subtotal(nv(_, K, A, M), [K, A], [S = sum(M)]),
        M <= 10.

/* A sa v K nikdy nevypil */
najviac10v(K, A) ←
        capuje(K, A),
        not nv2(K, A).

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

nv2(K, A) ←
        v(_, K, A, _).
```

Alebo:

```
answer(K, A) ←
        aspon200p(K),
        capuje(K, A), /* safety */
        not viac10v(K, A).

aspon200p(K) ←
        subtotal(n(K, P), [K], [C = count(P)]),
        C >= 200.

n(K, P) ←
        navstivil(_, P, K).

viac10v(K, A) ←
        subtotal(nv(_, K, A, M), [K, A], [S = sum(M)]),
        M > 10.

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

SQL:

```
create temporary table aspon200p as
select n.Krcma
from navstivil n
group by n.Krcma
having count(distinct n.Pijan) >= 200

create temporary table navstivil_vypil as
select n.Idn, n.Krcma, v.Alkohol, v.Mnozstvo /* Idn sa dá vynechať */
from navstivil n, vypil v
where n.Idn = v.Idn

create temporary table najviac10v as
(select nv.Krcma, nv.Alkohol
from navstivil_vypil nv
group by nv.Krcma, nv.Alkohol
having sum(nv.Mnozstvo) <= 10
)
union
(select c.Krcma, c.Alkohol
from capuje c
where not exists (
        select *
        from navstivil_vypil nv
        where nv.Krcma = c.Krcma and nv.Alkohol = c.Alkohol)
)

/* answer */
select v10.Krcma, v10.Alkohol
from aspon200p p200, najviac10v v10
where p200.Krcma = v10.Krcma
```

Alebo:

```
create temporary table aspon200p as
select n.Krcma
from navstivil n
group by n.Krcma
having count(distinct n.Pijan) >= 200

create temporary table viac10v as
select nv.Krcma, nv.Alkohol
from navstivil_vypil nv
group by nv.Krcma, nv.Alkohol
having sum(nv.Mnozstvo) > 10

/* answer */
select p200.Krcma, c.Alkohol
from aspon200p p200, capuje c
where p200.Krcma = v10.Krcma and not exists (
        select *
        from viac10v v10
        where c.Krcma = p200.Krcma and c.Alkohol = c.Alkohol)
```

2. a) Uveďte definíciu minimálneho pokrytia množiny funkčných závislostí. (Pojem pokrytia množiny funkčných závislostí nemusíte definovať, ten môžete použiť.) **(2)**

Minimálne pokrytie množiny funkčných závislostí F je také pokrytie G, že G obsahuje len kánonické funkčné závislosti (s 1 atribútom na pravej strane); a zároveň po vynechaní ľubovoľného atribútu z ľavej strany ľubovoľnej funkčnej závislosti z G množina G prestane byť pokrytím F; a zároveň po vynechaní ľubovoľnej funkčnej závislosti z G množina G prestane byť pokrytím F.

b) Algoritmus dekompozície relačnej schémy [r, F] do tretej normálnej formy obsahuje krok „Ak žiadna z relácií dekompozície $r_1, \ldots, r_N$ neobsahuje kľúč relácie r, tak pridaj k dekompozícii $r_1, \ldots, r_N$ reláciu s nejakým kľúčom relácie r“. Napíšte algoritmy (stačí pseudokód, resp. ideu) s polynomiálnou časovou zložitosťou, ktoré tento krok realizujú. (Treba vyriešiť dva problémy: 1.testovanie tej podmienky; 2.nájdenie nejakého kľúča relácie r.) **(2)**

1.Test, či niektorá z relácií $r_1, \ldots, r_N$ obsahuje nejaký kľúč:

```
for (i=1; i<N; i++)
{
        if ({r_i}^+ = r) /* počíta sa uzáver atribútov */
                return TRUE;  /* výpočet ďalej nepokračuje */
}
return FALSE;
```

2.Nájdenie nejakého kľúča relácie r:
Minimalizuj ľavú stranu (platnej) funkčnej závislosti $A_1...A_m \rightarrow A_1...A_m$ (kde A_1, ..., A_m sú všetky atribúty r) vzhľadom na množinu funkčných závislostí $F \cup A_1...A_m \rightarrow A_1...A_m$, tak ako v druhom kroku algoritmu pre konštrukciu (nejakého) minimálneho pokrytia F. Na poradí odstraňovania atribútov nezáleží. Minimalizovaná ľavá strana je (nejakým) kľúčom relácie r.

Výpočet uzáveru atribútov a minimalizácia ľavej strany funkčnej závislosti sú algoritmy s polynomiálnou časovou zložitosťou.

c) Existuje algoritmus, ktorý ľubovoľnú relačnú schému dekomponuje do Boyce-Coddovej normálnej formy tak, že výsledná dekompozícia je bezstratová (spája sa bezstratovo) a zachováva všetky funkčné závislosti? Svoju odpoveď ÁNO resp. NIE zdôvodnite. **(2)**

Taký algoritmus (bohužiaľ) neexistuje, lebo v niektorých prípadoch neexistuje ani dekompozícia, ktorá spĺňa tieto požiadavky. Napríklad, relačná schéma [r(A, B, C), {AB→C; C→A}] nie je v BCNF kvôli funkčnej závislosti C→A (C nie je nadkľúč r). Akákoľvek dekompozícia zlomí funkčnú závislosť AB→C.

d) Aký negatívny dôsledok má v databázovej aplikácii použitie dekompozície, ktorá láme (t.j. nezachováva) niektoré funkčné závislosti? **(2)**

Zlomenie funkčnej závislosti komplikuje automatickú kontrolu konzistencie databázy pri aktualizácii. Integritnú podmienku (constraint) zodpovedajúcu zlomenej funkčnej závislosti nie je možné vyjadriť lokálne v rámci jednej relácie (pri *create table*).

3. Daný je datalogový program
s(X, Y) ← c(X, _), c(_, Y), not q(X, Y).
s(X, Y) ← c(X, Y), not q(X, Y).
q(X, Y) ← c(X, _), c(_, Y), not c(X, Y).
q(X, Y) ← c(X, Y), q(Y, X).
a) Zapíšte výpočet dotazu ?- s(X, Y) pre daný program v relačnej algebre. **(2)**

Univerzálnou metódou výpočtu datalogového programu je (semi-) naivná evaluácia. Ale čiastočné pochopenie programu môže výpočet uľahčiť. Napríklad, druhé pravidlo pre s nevypočíta oproti prvému pravidlu žiadnu ďalšiu dvojicu [X, Y]. To prvé pravidlo je všeobecnejšie, o čom sa dá presvedčiť tak, že za prvú anonymnú premennú dosadíme Y a za druhú dosadíme X. Teda druhé pravidlo môžeme ignorovať a uvažovať program
r1: s(X, Y) ← c(X, _), c(_, Y), not q(X, Y).
r2: q(X, Y) ← c(X, _), c(_, Y), not c(X, Y).
r3: q(X, Y) ← c(X, Y), q(Y, X).

Idea plánu výpočtu: raz aplikovať r2 (telo pravidla r2 neobsahuje žiadne intenzionálne predikáty); potom iterovať r3 kým do relácie q pribúdajú nové dvojice; potom raz aplikovať pravidlo r1.

V relačnej algebre (postupnosť priradení):
$q := (\pi_X(c) \times \pi_Y(c)) - c$
$\varnothing(q := q \cup (c \bowtie P_{q(Y,X)}(q(X, Y))))$ /* iteruj, kým sa q mení */
$s := (\pi_X(c) \times \pi_Y(c)) - q$ /* relácia s obsahuje výsledok dotazu */

Pre daný program stačí jedna iterácia ∅ pre výpočet výslednej relácie (to platí pre ľubovoľné naplnenie extenzionálnej relácie c), čo umožňuje ďalšiu optimalizáciu výpočtu. Taktiež výsledok spoločného podvýrazu $\pi_X(c) \times \pi_Y(c)$ stačí vypočítať len raz. Ďalej, do výsledku joinu pri výpočte q podľa pravidla r3 netreba ukladať dvojice, ktoré boli vypočítané podľa pravidla r2 (podobne ako pri seminaivnej evaluácii). Optimalizovaný plán:
$cxc := \pi_X(c) \times \pi_Y(c)$
$q := cxc - c$
$q := q \cup ((c \bowtie P_{q(Y,X)}(q(X, Y))) - q)$
$s := cxc - q$ /* relácia s obsahuje výsledok dotazu */

b) Vypočítajte výsledok dotazu ?- s(X, Y) pre daný program s extenzionálnu databázou
c={[1, 1], [1, 2], [2, 2], [2, 3], [3, 1], [3, 3], [4, 1], [5, 6]}. **(2)**

V tele pravidla r2 nie sú žiadne intenzionálne predikáty, preto ho stačí vypočítať len raz:
$q := (\pi_X(c) \times \pi_Y(c)) - c$
q(X, Y) = {[1, 3], [1, 6], [2, 1], [2, 6], [3, 2], [3, 6], [4, 2], [4, 3], [4, 6], [5, 1], [5, 2], [5, 3]}

Ďalej budeme iterovať pravidlo r3, kým do relácie q pribúdajú nové dvojice:

$\varnothing(q := q \cup (c \bowtie P_{q(Y,X)}(q(X, Y))))$ /* iteruj, kým sa q mení */
q(X, Y) = **{[1, 2]**, [1, 3], [1, 6], [2, 1], **[2, 3]**, [2, 6], **[3, 1]**, [3, 2], [3, 6], [4, 2], [4, 3], [4, 6], [5, 1], [5, 2], [5, 3]}
(v druhej iterácii už nepribudnú žiadne dvojice)

Pravidlo r1 stačí vypočítať raz:
$s := (\pi_X(c) \times \pi_Y(c)) - q$ /* relácia s obsahuje výsledok dotazu */
s(X, Y) = {[1, 1], [2, 2], [3, 3], [4, 1], [5, 6]}

4. Uveďte príklad rozvrhu, v ktorom všetky transakcie končia commitom, a ktorý:
a) je konflikt-sériovateľný a nedá sa generovať dvojfázovým zamykaním; **(1)**

w1(X), r2(X), r1(X), c1, c2

Tento rozvrh je sériovateľný, konflikt-ekvivalentný sériovému rozvrhu T1→T2.
Pri dvojfázovom zamykaní (uvažujeme jednoduchú verziu 2PL) transakcia T1 nesmie odovzdať svoj write-lock na X až kým sa nevykoná r1(X). Tým pádom nemôže T2 medzi w1(X) a r1(X) držať konfliktný zámok na X.

b) je striktný a nie je konflikt-sériovateľný; **(1)**

r1(X), w2(X), r2(Y), w1(Y), c1, c2

Tento rozvrh neobsahuje dirty read ani dirty write, takže je striktný.
Nie je však konflikt-sériovateľný kvôli poradiu konfliktov r1(X)→w2(X) a r2(Y)→w1(Y).

c) je striktný a je konflikt-sériovateľný a nie je view-sériovateľný; **(1)**

Taký rozvrh neexistuje, lebo každý konflikt-sériovateľný rozvrh je view-sériovateľný.

d) je konflikt-sériovateľný a je obnoviteľný a nevyhýba sa kaskádovým abortom. **(1)**

w1(X), r2(X), c1, c2

Tento rozvrh je konflikt-sériovateľný, konflikt-ekvivalentný sériovému rozvrhu T1→T2.
Nevyhýba sa kaskádovým abortom kvôli dirty read w1(X)→r2(X).
Obnoviteľný je, lebo poradie commitov c1→c2 je rovnaké ako poradie operácií v dirty read.

e) je view-sériovateľný a nie je konflikt-sériovateľný a nie je obnoviteľný. **(1)**

w1(X), r2(X), w2(Y), w1(Y), w2(Y), c2, c1

Tento rozvrh je view-sériovateľný, view-ekvivalentný sériovému rozvrhu T1→T2. Nie je konflikt-sériovateľný, kvôli cyklu v precedenčnom-grafe T2→T1→T2. Nie je obnoviteľný kvôli dirty read w1(X)→r2(X) a poradiu commitov c2→c1.

Zdôvodnite prečo Váš rozvrh má požadované vlastnosti.