11/1/2022 Úvod do databáz, skúškový test, max **60** bodov

1. Uvažujte databázu 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) Sformulujte bezpečný dotaz v Datalogu **(6)** a relačnej algebre **(6)** na krčmy K, ktoré navštívil pijan, ktorý vypil aspoň pivo alebo rum pri každej svojej návšteve K.

Datalog:

```
answer(K) ←
        navstivil(_, P, K),
        not niekedy_nevypil(P, K).

niekedy_nevypil(P, K) ←
        navstivil(I, P, K),
        not v(I, pivo).
        not v(I, rum).

v(I, A) ←
        vypil(I, A, _).
```

Relačná algebra:

niekedy_nevypil := $\pi_{Pijan,\ Krcma}$ ((navstivil ▷ $\sigma_{Alkohol\ =\ 'pivo'}$ (vypil)) ▷ $\sigma_{Alkohol\ =\ 'rum'}$ (vypil))
/* answer */
π_{Krcma} (navstivil ▷ niekedy_nevypil)

b) Sformulujte bezpečný dotaz v Datalogu **(6)** a SQL **(6)** na trojice [P, K, X], kde X je priemerný počet druhov alkoholu, ktoré pijan P vypil počas jednej návštevy krčmy K. Priemer X zahŕňa aj tie návštevy P v K, pri ktorých P nevypil nič.

Datalog:

```
answer(P, K, X) ←
        subtotal(pocet(_, P, K, C), [P, K], [X = avg(C)]).

pocet(I, P, K, C) ←
        navstivil(I, P, K),
        subtotal(vypil(I, A, _), [I], [C = count(A)]).

pocet(I, P, K, 0) ←
        navstivil(I, P, K),
        not v(I).

v(I) ←
        vypil(I, _, _).
```

SQL:

```
with
pocet as
(
(
select n.Idn, n.Pijan, n.Krcma, count(v.Alkohol) as C
from navstivil n, vypil v
where n.Idn = v.Idn
group by n.Idn, n.Pijan, n.Krcma
)
union
(
select n.Idn, n.Pijan, n.Krcma, 0 as C
from navstivil n
where not exists (
        select *
        from vypil v
        where n.Idn = v.Idn
)
)
select p.Pijan, p.Krcma, avg(C) as X
from pocet p
group by p.Pijan, p.Krcma
```

2. a) Uveďte definíciu minimálneho pokrytia množiny funkčných závislostí. **(6)**

Množina funkčných závislostí M je minimálnym pokrytím množiny funkčných závislostí F, ak M obsahuje len kánonické funkčné závislosti (s najviac 1 atribútom na pravej strane); $M^+ = F^+$ (M pokrýva F a naopak, t.j. ich uzávery sú rovnaké); pre každú funkčnú závislosť z M platí, že po vynechaní ľubovoľného atribútu z jej ľavej strany alebo po vynechaní tej funkčnej závislosti z M prestane platiť $M^+ = F^+$.

b) Algoritmus dekompozície relačnej schémy [r, F] do tretej normálnej formy obsahuje krok „Ak žiadna z relácií dekompozície $r_1, ..., r_N$ neobsahuje kľúč relácie r, tak pridaj k dekompozícii $r_1, ..., r_N$ reláciu s nejakým kľúčom relácie r“. Napíšte algoritmy (stačí dostatočne zrozumiteľný pseudokód) s polynomiálnou časovou zložitosťou, ktoré tento krok realizujú. (Treba vyriešiť dva problémy: 1.testovanie tej podmienky; 2.nájdenie nejakého kľúča relácie r.) Vysvetlite. **(6)**

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

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

2. Nájdenie nejakého kľúča relácie r:
Minimalizuj ľavú stranu (platnej) funkčnej závislosti $A_1 \dots A_m \rightarrow A_1 \dots A_m$ (kde $A_1 \dots A_m$ sú všetky atribúty r) vzhľadom na množinu funkčných závislostí F, 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, lebo je nadkľúčom r a žiadna jej vlastná podmnožina nie je nadkľúčom r.

Výpočet uzáveru atribútov aj minimalizácia ľavej strany funkčnej závislosti sú algoritmy s polynomiálnou časovou zložitosťou (vzhľadom na počet atribútov r a funkčných závislostí v F).

3. Uvažujte SQL dotaz nad reláciou r(X, Y, Z) bez duplikátov a null hodnôt:
select distinct r1.X, sum(r1.Y) as S, sum(distinct r1.Y) as D
from r r1 group by r1.X having not (sum(r1.Y) = sum(distinct r1.Y))
a) Uveďte príklad naplnenia r, pre ktoré daný dotaz vráti presne jednu trojicu. Uveďte trojicu, ktorá je výsledkom dotazu. **(6)**

Napríklad pre r(X, Y, Z) = {[0, 1, 1], [0, 1, 2]} je výsledkom dotazu {[0, 2, 1]}.

b) Sformulujte daný dotaz ekvivalentne v Datalogu. **(6)**

```
answer(X, S, D) ←
        subtotal(r(X, Y, _), [X], [S = sum(Y)]),
        subtotal(r2(X, Y), [X], [D = sum(Y)]),
        not S = D.

r2(X, Y) ←
        r(X, Y, _).
```

4. Uveďte definíciu obnoviteľného rozvrhu.

Rozvrh je obnoviteľný, ak pre každý dirty read, kde transakcia TR číta nepotvrdené dáta od inej transakcie TW (t.j. TW pred tým čítaním tie dáta modifikovala a v momente toho čítania nie je commitovaná), platí, že commitu TR predchádza commit TW.

a) Uvažujte modifikáciu striktného dvojfázového zamykania, v ktorej ľubovoľná transakcia smie postupne odovzdávať zámky na čítanie (ktoré drží) kedykoľvek počas svojej druhej fázy. Zámky na písanie (ktoré drží) smie odovzdať až pri žiadosti o commit (alebo pri aborte). Rozhodnite, či takto modifikovaná metóda garantuje, že výstupný rozvrh je konflikt-sériovateľný a striktný. Odpoveď ÁNO resp. NIE zdôvodnite (dokážte alebo uveďte kontrapríklad). **(6)**

Keďže táto metóda rešpektuje pravidlá dvojfázového zamykania (dokonca je ešte reštriktívnejšia), výstupný rozvrh je konflikt-sériovateľný.

Sporom ukážeme, že pri tejto metóde nebude vo výstupnom rozvrhu dirty read ani dirty write. Predpokladajme, že výstupný rozvrh obsahuje dirty read (pre dirty write je dôvodenie analogické) a bez ujmy na všeobecnosti predpokladajme, že to je prvý dirty read vo výstupnom rozvrhu. To znamená, že nejaká transakcia TR v tej chvíli číta (s prideleným read lock) objekt, ktorý bol predtým modifikovaný transakciou TW, pričom TW je aktívna v momente toho dirty read (inak by to nebol dirty read). Lenže na modifikáciu toho objektu potrebovala transakcia TW write lock, ktorý musela neskôr odovzdať (na to, aby TR získala read lock). To je však v spore s tým, že pri danej metóde TW odovzdáva write locks až pri svojom commite či aborte, a vtedy už aktívna nie je.

ÁNO. výstupný rozvrh bude konflikt-sériovateľný a striktný.

b) Uvažujte modifikáciu striktného dvojfázového zamykania, v ktorej ľubovoľná transakcia smie postupne odovzdávať zámky na písanie (ktoré drží) kedykoľvek počas svojej druhej fázy. Zámky na čítanie (ktoré drží) smie odovzdať až pri žiadosti o commit (alebo pri aborte). Rozhodnite, či takto modifikovaná metóda garantuje, že výstupný rozvrh je konflikt-sériovateľný a obnoviteľný. Odpoveď ÁNO resp. NIE zdôvodnite (dokážte alebo uveďte kontrapríklad). **(6)**

Keďže táto metóda rešpektuje pravidlá dvojfázového zamykania (dokonca je ešte reštriktívnejšia), výstupný rozvrh je konflikt-sériovateľný.

Výstupný rozvrh však môže byť takýto:
s1, s2, wl1(X), w1(X), ul1(X), rl2(X), r2(X), wl2(Y), w2(Y), ul2(Y), ul2(X), c2
(pričom tie dve posledné operácie systém vykoná atomicky).
Z výstupného rozvrhu môžeme vynechať prideľovanie a odovzdávanie zámkov (tieto operácie systém vykonáva len interne):
s1, s2, w1(X), r2(X), w2(Y), c2
r2(X) je dirty read, c2 predchádza ukončeniu T1. Tento rozvrh nie je obnoviteľný, lebo r2(X) je dirty read (T2 číta nepotvrdenú hodnotu od T1) a T2 commituje pred T1.

NIE. Výstupný rozvrh bude konflikt-sériovateľný, ale nemusí byť obnoviteľný.