6/2/2018 Úvod do databáz, skúškový test, max **60** bodov

1. Uvažujte databázu bez duplikátov a null hodnôt: capuje(Krcma, Alkohol),
navstivil(Idn, Pijan, Krcma), vypil(Idn, Alkohol, Mnozstvo).
Platí: Idn → Pijan, Krcma; Idn, Alkohol → Mnozstvo; Mnozstvo > 0.
Hodnoty Idn sú časové pečiatky návštev, teda tvoria rastúcu postupnosť s rastúcim časom.
a) Sformulujte bezpečný dotaz v Datalogu **(6)** na také dvojice [K, A], že alkohol A vypil v krčme K každý pijan, ktorý K niekedy navštívil; ale žiaden pijan nevypil alkohol A v krčme K dvakrát za sebou (t.j. počas svojich dvoch za sebou nasledujúcich návštev krčmy K).

```
answer(K, A) ←
        capuje(K, A),
        not niekto_nikdy_nevypil(K, A),
        not niekto_vypil_nasledne(K, A).

niekto_nikdy_nevypil(K, A) ←
        capuje(K, A),
        navstivil(_, P, K),
        not niekedy_vypil(P, K, A).

niekedy_vypil(P, K, A) ←
        navstivil(I, P, K),
        vypil(I, A, _).

niekto_vypil_nasledne(K, A) ←
        navstivil(I1, P, K),
        vypil(I1, A, _),
        navstivil(I2, P, K),
        vypil(I2, A, _),
        not navsteva_medzi(I1, I2).

navsteva_medzi(I1, I2) ←
        navstivil(I1, P, K),
        navstivil(I2, P, K),
        navstivil(I, P, K),
        I1 < I,
        I < I2.
```

b) Sformulujte bezpečný dotaz v relačnom kalkule **(6)**, Datalogu **(6)** a SQL **(6)** na trojice [K1, K2, A] pre ktoré platí, že v krčme K1 sa celkovo vypilo väčšie množstvo alkoholu A než v krčme K2, ale alkohol A vypilo v krčme K1 menej pijanov než v krčme K2.

Relačný kalkul:

```
{[K1, K2, A]:
        ∃S1 ∃S2 ∃C1 ∃C2
        ♥I, S1 = sum(M) (∃P navstivil(I, P, K1) ∧ vypil(I, A, M)) ∧
        ♥I, S2 = sum(M) (∃P navstivil(I, P, K2) ∧ vypil(I, A, M)) ∧
        ♥C1 = count(P) (∃I ∃M navstivil(I, P, K1) ∧ vypil(I, A, M)) ∧
        ♥C2 = count(P) (∃I ∃M navstivil(I, P, K2) ∧ vypil(I, A, M)) ∧
        S1 > S2 ∧
        C1 < C2
}
```

Datalog:

```
answer(K1, K2, A) ←
        subtotal(mnozstva(_, K1, A, M), [K1, A], [S1 = sum(M)]),
        subtotal(mnozstva(_, K2, A, M), [K2, A], [S2 = sum(M)]),
        subtotal(pocty(K1, A, P), [K1, A], [C1 = count(P)]),
        subtotal(pocty(K2, A, P), [K2, A], [C2 = count(P)]),
        S1 > S2,
        C1 < C2.

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

pocty(K, A, P) ←
        navstivil(I, P, K),
        vypil(I, A, _).
```

SQL:

```
with mnozstva_pocty as
(
        select n.Krcma, v.Alkohol, sum(v.Mnozstvo) as S, count(distinct n.Pijan) as C
        from navstivil n, vypil v
        where n.Idn = v.Idn
        group by n.Krcma, v.Alkohol
)
select mp1.Krcma, mp2.Krcma, mp1.Alkohol
from mnozstva_pocty mp1, mnozstva_pocty mp2
where mp1.Alkohol = mp2.Alkohol and mp1.S > mp2.S and mp.mp1.C < mp2.C
```

2. Daná je relácia r(A, B, C, D, E, F, G, H) s funkčnými závislosťami
A→E, AC→BDE, AD→B, BCD→GH, BCDF→A, BE→D, BG→F, BH→E, E→B.
a) Dekomponujte r do tretej normálnej formy, bezstratovo a so zachovaním všetkých funkčných závislostí. **(6)**

Vypočítajme nejaké minimálne pokrytie danej množiny funkčných závislostí. Po minimalizácii ľavých strán kánonických funkčných závislostí:
A→E, A→B, A→D, A→E, A→B, BCD→G, BCD→H, BCD→A, BG→F, BH→E, E→B, E→D.
Minimálne pokrytie (po odstránení redundantných funkčných závislostí):
A→E, BCD→G, BCD→H, BCD→A, BG→F, BH→E, E→B, E→D.

3NF dekompozícia z minimálneho pokrytia spĺňa uvedené požiadavky:
(A, E)
(A, B, C, D, G, H) /* nadkľúč v r */
(B, D, E)
(B, E, H)
(B, F, G)

b) Dekomponujte r do Boyce-Coddovej normálnej formy, bezstratovo. Snažte sa vyhnúť zlomeniu funkčných závislostí. **(6)**

Začnime s 3NF dekompozíciou z úlohy a).
(A, E) je v BCNF.
(A, B, C, D, G, H) nie je v BCNF, lebo platí A→B, ale neplatí napr. A→G. Dekomponujeme do (A, B) a (A, C, D, G, H).
(A, B) je v BCNF.
(A, C, D, G, H) nie je v BCNF, lebo platí A→D, ale neplatí napr. A→G. Dekomponujeme do (A, D) a (A, C, G, H).
(A, D) je v BCNF.
(A, C, G, H) je v BCNF.
(B, D, E) je v BCNF.
(B, E, H) nie je v BCNF, lebo platí E→B, ale neplatí E→H. Dekomponujeme do (B, E) a (E, H).
(B, E) je v BCNF.
(E, H) je v BCNF.
(B, F, G) je v BCNF.

Výsledná BCNF dekompozícia:
(A, B) /* A→B */
(A, C, G, H)
(A, D) /* A→D */
(A, E) /* A→E */
(B, D, E) /* E→B, E→D */
(B, F, G) /* BG→F */
(E, H)

3. a) Definujte pojem bezpečnosti Datalogových programov. Vysvetlite prečo je tento pojem dôležitý v kontexte databázových systémov. Uveďte príklad Datalogového programu, ktorý nie je bezpečný, vysvetlite dôsledky. **(6)**

Definícia:
Datalogový program je bezpečný, keď každé jeho pravidlo je bezpečné.
Datalogové pravidlo je bezpečné, keď každá premenná, ktorá sa v ňom vyskytuje, sa nachádza v tele toho pravidla v niektorom relačnom podcieli, ktorý nie je negovaný.
Relačný podcieľ je extenzionálny databázový predikát alebo predikát, ktorý je definovaný tým programom (t.j. je v hlave niektorého pravidla).

Prečo je tento pojem dôležitý pre databázové systémy:
Chceme, aby výsledok akéhokoľvek dotazu na daný program bol doménovo nezávislý, t.j. aby nezávisel od typov atribútov, ale len od naplnenia databázy. Ak je program bezpečný, tak typy atribútov neovplyvňujú výsledky dotazov.
Chceme vedieť ľubovoľný Datalogový program počítať „mechanicky“, t.j. vždy tým istým algoritmom. Ak je program bezpečný, tak sa dá počítať naivnou evaluáciou.
Horeuvedená definícia umožňuje algoritmicky rozhodnúť (overiť jednoduchou syntaktickou analýzou, pravidlo po pravidle) či je program bezpečný.

Sú programy, ktoré nie sú bezpečné, ale napriek tomu sú doménovo nezávislé a vieme ich počítať. Avšak nie všetky sú také. Napríklad
p(_).
je program, ktorý nie je bezpečný, lebo anonymná premenná v hlave pravidla nie je použitá v pozitívnom relačnom podcieli toho pravidla. Tento program definuje predikát p(.), ktorý je pravdivý pre „čokoľvek“. Lenže výsledok dotazu
?- p(X).
závisí od typu argumentu p(.) (typ môže byť nekonečná, možno nespočítateľná množina).

b) Definujte pojem stratifikácie Datalogových programov. Vysvetlite prečo je tento pojem dôležitý v kontexte databázových systémov. Uveďte príklad bezpečného Datalogového programu, ktorý nie je stratifikovateľný, vysvetlite dôsledky. **(6)**

Definícia:
Datalogový program ktorý definuje predikáty $p_1, ..., p_N$ je stratifikovaný, keď každému p_i je pridelené prirodzené číslo (stratum) $s(p_i)$ tak, že platí:
Ak sa v tele pravidla s hlavou p_i vyskytuje pozitívny podcieľ p_j, tak $s(p_i) \geq s(p_j)$.
Ak sa v tele pravidla s hlavou p_i vyskytuje negovaný podcieľ p_j, tak $s(p_i) > s(p_j)$.
Datalogový program je stratifikovateľný, ak preň existuje očíslovanie predikátov s horeuvedenou vlastnosťou.

Dá sa algoritmicky (v polynomiálnom čase) rozhodnúť, či program je stratifikovateľný. V prípade programu s N predikátmi stačí uvažovať strata od 1 po N. (Extenzionálne predikáty majú stratum 1.)

Prečo je tento pojem dôležitý pre databázové systémy:
Chceme, aby výpočet programov (resp. akéhokoľvek dotazu na akýkoľvek program, pre akékoľvek naplnenie extenzionálnej databázy) skončil. Lenže každý „dostatočne silný“ programovací jazyk umožňuje písať programy, ktoré nie vždy skončia.
Pre všetky stratifikovateľné Datalogové programy je ukončenie výpočtu naivnou evaluáciou garantované.
Dôsledkom je, že výpočty dotazov na programy bez negácie (aj rekurzívne) vždy skončia.

Pre nestratifikovateľné programy ukončenie naivnej evaluácie garantované nie je. Napríklad:
p(X) ← r(X), not p(X).
je bezpečný program, ale nie je stratifikovateľný. Pre extenzionálnu databázu r(.) = {1} naivná evaluácia neskončí, lebo po párnych iteráciách bude p(.) = {}, po nepárnych iteráciách bude p(.) = {1}.

4. Relácia *data(X, Y, Z)* je uložená na disku v 100 000 blokoch. Bloky v RAM a na disku sú rovnako veľké. Stredná doba prenosu diskového bloku (do RAM aj opačne) je 0.1 s. Na výpočet dotazu *select X, Y, Z from data order by X, Y* sa použije externý merge-sort.
a) Určite čas vykonávania dotazu, ak v RAM je rezervovaných 200 blokov. Zdôvodnite. **(6)**

Poďme počítať vstupno-výstupné operácie pre merge-sort s 200 blokmi.

V 1. fáze merge-sort sa prečíta prvých 200 blokov data do RAM, utriedi a zapíše do behu. Toto sa zopakuje kým sa nespracuje celá relácia data, t.j. 500 krát (= 100000 / 200). Vznikne 500 utriedených behov veľkosti 200 blokov. Keďže každý blok data sa raz prečíta a raz zapíše, dokopy sa urobí 100000 read operácií a 100000 write operácií.

V 2. fáze (merge) sa 199 blokov v RAM použije na zlučovanie niektorých 199 behov, 1 blok sa použije na zápis zlúčeného behu veľkosti 39800 blokov. Urobí sa 39800 (= 199 * 200) read operácií a 39800 (= 199 * 200) write operácií.

2. fáza pokračuje zlučovaním ďalších 199 behov veľkosti 200 blokov tak, ako v predošlom odstavci. Urobí sa 39800 (= 199 * 200) read operácií a 39800 (= 199 * 200) write operácií.

2. fáza pokračuje zlučovaním 102 behov veľkosti 200 blokov a 2 behov veľkosti 39800 blokov (ktoré vznikli predošlými zlučovaniami). Tým sa zlúčia všetky zostávajúce behy do jedného behu veľkosti 100000 blokov. Urobí sa 100000 read operácií a 100000 write operácií.

Dokopy sa urobí 279600 read a 279600 write operácií, teda 559200 vstupno-výstupných operácií. Ak každá z nich trvá 0.1 s, **dolný odhad na čas vykonávania je 55920 sekúnd, t.j. ca. 15.5 hodín**. Zanedbávame réžiu spracovania záznamov v RAM. Niekoľko blokov uložených v RAM vieme občas možno „recyklovať“, t.j. nemusíme ich opätovne čítať. Toto však čas vykonávania výrazne nezníži.

b) Určite čas vykonávania dotazu, ak v RAM je rezervovaných 2000 blokov. Zdôvodnite. **(6)**

V 1. fáze merge-sort sa prečíta prvých 2000 blokov data do RAM, utriedi a zapíše do behu. Toto sa zopakuje kým sa nespracuje celá relácia data, t.j. 500 krát (= 100000 / 200). Vznikne 50 utriedených behov veľkosti 2000 blokov. Keďže každý blok data sa raz prečíta a raz zapíše, dokopy sa urobí 100000 read operácií a 100000 write operácií.

V 2. fáze (merge) sa 50 blokov v RAM použije na súčasné zlúčenie všetkých behov, 1 blok sa použije na zápis zlúčeného behu. Urobí sa 100000 read operácií a 100000 write operácií.

Dokopy sa urobí 200000 read operácií a 200000 write operácií, či je 400000 vstupno-výstupných operácií. Ak každá z nich trvá 0.1 s, **dolný odhad na čas vykonávania je 40000 sekúnd, t.j. ca. 11 hodín**. Zanedbávame réžiu spracovania záznamov v RAM. Toto však čas vykonávania výrazne nezníži.