13/1/2015 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 dvojice [K, A] také, že krčma K čapuje alkohol A; a zároveň každý pijan, ktorý ľúbi alkohol A, ho vypil pri niektorej návšteve krčmy K. Sformulujte tento dotaz v relačnom kalkule **(2)**, Datalogu **(2)** a relačnej algebre **(2)**.

Relačný kalkul:

```
{[K, A]:
        capuje(K, A) ∧
        ¬ /* zly_pijan(K, A) */
        (∃P
                capuje(K, A) ∧
                lubi(P, A) ∧
                ¬ /* niekedy_vypil(P, K, A) */
                (∃I ∃M
                        navstivil(I, P, K) ∧
                        vypil(I, A, M)
                )
        )
}
```

Datalog:

```
answer(K, A) ←
        capuje(K, A),
        not zly_pijan(K, A).

zly_pijan(K, A) ←
        capuje(K, A), /* safety */
        lubi(P, A),
        not niekedy_vypil(P, K, A).

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

Relačná algebra:

$niekedy\_vypil := \pi_{Pijan,\ Krcma,\ Alkohol}\ (navstivil \bowtie vypil)$

$zly\_pijan := \pi_{Krcma,\ Alkohol}\ ((capuje \bowtie lubi) - niekedy\_vypil)$

$answer := capuje - zly\_pijan$

b) Nájdite dvojice [P, A] také, že vo viacej než polovici krčiem, ktoré čapujú alkohol A, pijan P celkovo vypil viacej alkoholu A než ktorýkoľvek iný pijan. Sformulujte tento dotaz v Datalogu **(2)** a SQL **(2)**.

Datalog:

```
answer(P, A) ←
        subtotal(vitaz(P, A, K), [P, A], [CV = count(K)]),
        subtotal(capuje(K, A), [A], [C = count(K)]),
        2 * CV > C. /* CV2 is 2 * CV, CV2 > C */

vitaz(P, A, K) ←
        vypil_total(P, A, K, _),
        not niekto_iny_aspon_tolko(P, A, K).

vypil_total(P, A, K, T) ←
        subtotal(nv(P, A, K, M, _), [P, A, K], [T = sum(M)]).

niekto_iny_aspon_tolko(P, A, K) ←
        vypil_total(P, A, K, T),
        vypil_total(P2, A, K, T2),
        not P = P2,
        T2 >= T.

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

SQL:

```
create temporary table vypil_total as
select n.Pijan, v.Alkohol, n.Krcma, sum(v.Mnozstvo) as T
from navstivil n, vypil v
where n.Idn = v.Idn
group by n.Pijan, v.Alkohol, n.Krcma

create temporary table vitaz as
select vt.Pijan, vt.Alkohol, vt.Krcma
from vypil_total vt
where not exists ( /* niekto_iny_aspon_tolko(P, A, K) */
        select *
        from vypil_total vt2
        where vt2.Pijan <> vt.Pijan and vt2.Alkohol = vt.Alkohol and vt2.Krcma = vt.Krcma
        and vt2.T > vt.T
        )

create temporary table krcmy_vitazne as
select v.Pijan, v.Alkohol, count(distinct v.Krcma) as C
from vitaz v
group by v.Pijan, v.Alkohol

create temporary table krcmy_capujuce as
select c.Alkohol, count(distinct v.Krcma) as C
from capuje c
group by c.Alkohol

/* answer */
select kv.Pijan, kv.Alkohol
from krcmy_vitazne kv, krcmy_capujuce kc
where kv.Alkohol = kc.Alkohol and 2 * kv.C > kc.C
```

2. Daná je relácia r(A, B, C, D, E, F, G) s funkčnými závislosťami
ABCD → EF; ABE → FG; ABDG → CF; G → BD.
a) Nájdite všetky kľúče relácie r. **(2)**

ABCDEFG
+G: **AG**
–G: ABCDEF
–B: ACDEF
+B: ABCDEF
+E: **ABE**
–E: ABCDF
–C: ABDF
+C: ABCDF
–D: ABCF
+D: ABCDF
–F: **ABCD**

Kľúče sú ABCD, ABE, AG. Iné kľúče nie sú.

b) Dekomponujte r do tretej normálnej formy, bezstratovo a so zachovaním všetkých funkčných závislostí. **(2)**

Po minimalizácii ľavých strán kánonických funkčných závislostí:
ABCD → E
ABCD → F
ABE → F
ABE → G
AG → C
AG → F
G → B
G → D

Po odstránení redundantných funkčných závislostí dostávame (nejaké) minimálne pokrytie:
ABCD → E
ABE → G
AG → C
AG → F
G → B
G → D

3NF dekompozícia:
(A, B, C, D, E), (A, B, E, G), (A, C, F, G), (B, D, G)
Alebo:
(A, B, C, D, E), (A, B, E, G), (A, C, F, G), (D, G)

Alebo (všetky atribúty okrem F sú kľúčové):
(A, B, C, D, E, G), (A, F, G)

c) Uvažujte dekompozíciu r do (A, B, C, D, E), (A, C, F, G), (B, D, G). Rozhodnite, či je bezstratová (či sa spája bezstratovo) **(1)**;

	A	B	C	D	E	F	G
ABCDE	a1	a2	a3	a4	a5		
ACFG	a1		a3			a6	a7
BDG		a2		a4			a7

G → BD

	A	B	C	D	E	F	G
ABCDE	a1	a2	a3	a4	a5		
ACFG	a1	a2	a3	a4		a6	a7
BDG		a2		a4			a7

ABCD → E

	A	B	C	D	E	F	G
ABCDE	a1	a2	a3	a4	a5		
ACFG	a1	a2	a3	a4	a5	a6	a7
BDG		a2		a4			a7

ABE → FG

	A	B	C	D	E	F	G
ABCDE	**a1**	**a2**	**a3**	**a4**	**a5**	**a6**	**a7**
ACFG	a1	a2	a3	a4	a5	a6	a7
BDG		a2		a4			a7

Áno, daná dekompozícia je bezstratová.

či zachováva všetky funkčné závislosti **(1)**;

Stačí overiť zachovanie funkčných závislostí minimálneho pokrytia z podúlohy b). Zistíme, že funkčná závislosť ABE → G nie je zachovaná v žiadnej relácii danej dekompozície. **Teda nie je pravda, že daná dekompozícia zachováva všetky funkčné závislosti.**

či je v Boyce-Coddovej normálnej forme **(2)**.

V relácii (B, D, G) platia len netriviálne funkčné závislosti G → B a G → D, pričom G je nadkľúč tej relácie. Táto relácia je v BCNF.

V relácii (A, C, F, G) platia len netriviálne funkčné závislosti AG → C a AG → F, pričom AG je nadkľúč tej relácie. Táto relácia je v BCNF.

V relácii (A, B, C, D, E) platia len netriviálne funkčné závislosti ABCD → E, ABE → C, ABE → D, pričom ABCD a ABE sú nadkľúče tej relácie. Táto relácia je v BCNF.

Áno, daná dekompozícia je v BCNF.

3. Pri opätovnom štarte databázového systému obsahuje log-file nasledujúce záznamy (zoradené od najstaršieho po najnovší):
<T0, start>, <T0, A, 20, 100>, <T1, start>, <T0, commit>, <T2, start>, <T2, A, 100, 50>, <T2, B, 200, 250>, <T1, A, 50, 70>, <T1, commit>.
a) Odsimulujte algoritmus obnovy (bez predpokladov na cache) pre tento log-file, okomentujte jednotlivé kroky algoritmu. **(2)**

Zostupný prechod cez log (undo):

	UNDO_LIST := {}; REDO_LIST := {} /* inicializácia */
<T1, commit>:	REDO_LIST := {T1} /* pridaj T1 do REDO_LIST */
<T1, A, 50, 70>	
<T2, B, 200, 250>	UNDO_LIST := {T2}; B := 200 /* undo */
<T2, A, 100, 50>	A := 100 /* undo */
<T2, start>	UNDO_LIST := {} /* T2 netreba ďalej evidovať */
<T0, commit>	REDO_LIST := {T1, T0} /* pridaj T0 do REDO_LIST */
<T1, start>	
<T0, A, 20, 100>	
<T0, start>	

Vzostupný prechod cez log (redo):

<T0, start>	
<T0, A, 20, 100>	A := 100 /* redo */
<T1, start>	
<T0, commit>	REDO_LIST := {T1} /* T0 netreba ďalej evidovať */
<T2, start>	
<T2, A, 100, 50>	
<T2, B, 200, 250>	
<T1, A, 50, 70>	A := 70 /* redo */
<T1, commit>	REDO_LIST := {}

b) Uvažujte systém, ktorý pri vykonávaní operácie write okamžite zapisuje novú hodnotu do databázy. V ktorom momente je v takomto systéme potrebné vykonať operáciu UNDO resp. REDO počas normálnej prevádzky? Vysvetlite. **(1)**

Operáciu REDO netreba počas bežnej prevádzky nikdy vykonať.
Operáciu UNDO treba vykonať hneď po aborte niektorej transakcie.

4. Uvažujte SQL dotaz select r.X, s.X from r, s where r.X = s.X and r.Y > 7. Relácia r je uložená v 500 diskových blokoch, relácia s je uložená v 1000 diskových blokoch. Bloky na disku a v operačnej pamäti sú rovnako veľké. Relácie r, s neobsahujú duplikáty ani null hodnoty, atribút X je kľúčom v r aj v s. K dispozícii je 200 voľných blokov v operačnej pamäti.
a) Dotaz sa počíta metódou nested-loop-join. Popíšte spôsob využitia voľných blokov v RAM. **(1)**

198 blokov sa použije pre vstup relácie r
1 blok sa použije pre vstup relácie s
1 blok sa použije pre výstup výslednej relácie

Uveďte počet vstupných diskových operácií (z disku do RAM), vysvetlite. **(1)**

Prečíta sa 198 blokov r a postupne 1000 blokov s, pre každú kombináciu n-tíc r a s sa vypočíta výstup:
198 + 1000 = 1198 vstupných operácií

Prečíta sa ďalších 198 blokov r a postupne 999 blokov s (1 blok s je už v RAM), pre každú kombináciu n-tíc r a s sa vypočíta výstup:
198 + 999 = 1197 vstupných operácií

Prečíta sa posledných 104 blokov r a 999 blokov s (1 blok s je už v RAM), pre každú kombináciu n-tíc r a s sa vypočíta výstup:
104 + 999 = 1103 vstupných operácií

Celkový počet vstupných operácií: 1198 + 1197 + 1103 = **3498**

b) Uveďte vhodný spôsob indexovania relácií r a s a načrtnite metódu výpočtu daného dotazu, ktorá je vďaka indexovaniu efektívnejšia než nested-loop join. **(1)**

Pre výpočet tohto dotaz je vhodné vytvoriť B-tree (alebo B^+-tree) indexy pre relácie r a s podľa joinovacieho atribútu X. Usporiadanie r aj s podľa atribútu X umožňuje počítať daný dotaz metódou merge-join, ktorá funguje zhruba ako druhá fáza merge-sort. Keď vieme, že X je kľúčový atribút r aj s (teda pre ľubovoľnú hodnotu X existuje v r aj v s najviac 1 záznam), v RAM stačí dokonca rezervovať 1 blok pre vstup r, 1 blok pre vstup s a 1 blok pre výstup. Ideou je prečítať 1 blok z každej relácie a otestovať joinovaciu podmienku. Ak platí, do výstupného bloku sa zapíše výstupná dvojica. Ak neplatí, pokračuje sa v testovaní ďalším záznamom tej relácie, ktorej hodnota X je menšia než v tej druhej relácii (v prípade rovnosti hodnôt X sa prečíta nasledujúci záznam v r aj v s). Po prečítaní všetkých záznamov vo vstupnom bloku niektorej relácie sa prečíta z disku do RAM ďalší blok relácie s tou menšou hodnotou X. Testovanie joinovacej podmienky a „posúvanie menšej relácie“ sa opakuje až kým sa obe relácie nedočítajú.

Uveďte počet vstupných diskových operácií, vysvetlite. **(1)**

Pri použití merge-join sa každý blok r aj s prečíta najviac raz. Počet vstupných operácií je teda nanajvýš
500 + 1000 = 1500.

Namiesto vyhľadávacieho stromu sa dá použiť hashovaný index. Redukcia vstupných operácií môže byť dokonca ešte väčšia (s predpokladom perfektnej hashovacej funkcie).