14/1/2014 Úvod do databáz, **skúškový** test, max 25 bodov, 90 min

1. Daná je databáza (bez duplikátov a null hodnôt): capuje(Krcma, Alkohol, Cena),
lubi(Pijan, Alkohol), navstivil(Idn, Pijan, Krcma), vypil(Idn, Alkohol, Mnozstvo), produkuje(Alkohol, Firma).
Platí: Idn → Pijan, Krcma; Krcma, Alkohol → Cena;
Idn, Alkohol → Mnozstvo; Alkohol → Firma; Mnozstvo > 0; Cena > 0.
a) Sformulujte nasledujúci dotaz v rel. kalkule **(2)**, Datalogu **(2),** SQL **(2)** a rel. algebre **(2)**. Nájdite dvojice [P, A] také, že pijan P ľúbi alkohol A a ešte také dva ďalšie (navzájom rôzne) alkoholy, že pri každej návšteve krčmy, pri ktorej P vypil A, vypil aj niektorý z týchto dvoch ďalších alkoholov.

Relačný kalkul:

```
{[P, A]:
        ∃A1 ∃A2
        lubi(P, A) ∧ lubi(P, A1) ∧ lubi(P, A2) ∧ A1≠A2 ∧ A≠A1 ∧ A≠A2 ∧
        ¬ (
                ∃I ∃K ∃M
                navstivil(I, P, K) ∧ vypil(I, A, M) ∧
                ¬ (
                        ∃M1
                        vypil(I, A1, M1)
                ) ∧
                ¬ (
                        ∃M2
                        vypil(I, A2, M2)
                )
        )
}
```

Datalog:

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

v(I, A) ←
        vypil(I, A, _).

vynimka(P, A, A1, A2) ←
        lubi(P, A1), /* safety */
        lubi(P, A2), /* safety */
        navstivil(I, P, _),
        vypil(I, A, _),
        not v(I, A1),
        not v(I, A2).
```

SQL:

```
create temporary table vynimka as
select n.Pijan, v.Alkohol as A, l1.Alkohol as A1, l2.Alkohol as A2
from lubi l1, lubi l2, navstivil n, vypil v
where l1.Pijan = n.Pijan and l2.Pijan = n.Pijan and n.Idn = v.Idn and
        not exists (
                select *
                from vypil v1
                where v1.Idn = n.Idn and v1.Alkohol = l1.Alkohol)
        and not exists (
                select *
                from vypil v2
                where v2.Idn = n.Idn and v2.Alkohol = l2.Alkohol)

/* answer */
select l1.Pijan, l1.Alkohol
from lubi l1, lubi l2, lubi l3
where l1.Pijan = l2.Pijan and l1.Pijan = l3.Pijan and l1.Alkohol <> l2.Alkohol and l1.Alkohol <> l3.Alkohol and
        l2.Alkohol <> l3.Alkohol and not exists (
                select *
                from vynimka v
                where v.Pijan = l1.Pijan and v.A = l1.Alkohol and v.A1 = l2.Alkohol and v.A2 = l3.Alkohol)
```

Relačná algebra:

$vynimka\_pos := P_{l1(P, A1)}(lubi) \bowtie P_{l2(P, A2)}(lubi) \bowtie P_{n(I, P)}(\pi_{Idn, Pijan}(navstivil)) \bowtie P_{v(I, A)}(\pi_{Idn, Alkohol} vypil))$

$vynimka\_neg1 := (\pi_{I, A1}(vynimka\_pos) - P_{v(I, A1)}(\pi_{Idn, Alkohol} vypil))) \bowtie vynimka\_pos$

$vynimka\_neg2 := (\pi_{I, A2}(vynimka\_neg1) - P_{v(I, A2)}(\pi_{Idn, Alkohol} vypil))) \bowtie vynimka\_neg1$

$vynimka := \pi_{P, A, A1, A2}(vynimka\_neg1)$

$answer\_pos := \pi_{l1.P, l1.A, l2.A1, l3.A2}(\sigma_{l1.A \neq l2.A1 \wedge l1.A \neq l3.A2 \wedge l2.A1 \neq l3.A2}(P_{l1(P, A)}(lubi) \bowtie P_{l2(P, A1)}(lubi) \bowtie P_{l3(P, A2)}(lubi)))$

$answer := answer\_pos - vynimka$

b) Sformulujte nasledujúci dotaz v relačnom kalkule **(2)** a SQL **(2)**. Nájdite trojice [P, F, C], kde C je počet krčiem, v ktorých pijan P celkovo minul aspoň 50 EUR za alkoholy produkované firmou F. Vo výsledku nemajú byť trojice s C=0.

Relačný kalkul:

```
{[P, F, C]:
        ♥ C = count(K) (
                /* prepil50 */
                ∃S
                S >= 50 ∧
                ♥ I, A, S = sum(X) (
                        /* ucet */
                        ∃C ∃M
                        navstivil(I, P, K) ∧ vypil(I, A, M) ∧ capuje(K, A, C) ∧ produkuje(F, A) ∧ X = C * M
                )
        )
}
```

SQL:

```
create temporary table ucet as
select n.Idn, p.Firma, n.Pijan, n.Krcma, v.Alkohol, c.Cena * v.Mnozstvo as S
from navstivil n, vypil v, capuje c, produkuje p
where n.Idn = v.Idn and n.Krcma = c.Krcma and v.Alkohol = c.Alkohol and
        v.Alkohol = p.Alkohol

create temporary table prepil50 as
select U.Firma, U.Pijan, U.Krcma
from ucet U
group by U.Firma, U.Pijan, U.Krcma
having sum(U.S) >= 50

/* answer */
select p50.Pijan, p50.Firma, count(distinct p50.Krcma) as C
from prepil50 p50
group by p50.Pijan, p50.Firma
```

2. a) Definujte bezpečnosť Datalogových programov. **(1)**

Datalogový program je bezpečný, keď každé jeho pravidlo je bezpečné.
Datalogové pravidlo je bezpečné, keď každá premenná použitá kdekoľvek v tom pravidle sa vyskytuje (aj) v nejakom pozitívnom (t.j. nie negovanom) relačnom podcieli toho pravidla.

Relačný podcieľ je buď predikát extenzionálnej databázy alebo predikát definovaný niektorým pravidlom programu (t.j. predikát v hlave niektorého pravidla toho programu).
Bezpečné programy vieme počítať. Aritmetické podciele <,=, a pod. nepovažujeme za relačné podciele v horeuvedenom zmysle, lebo predstavujú relácie, ktoré vo všeobecnosti počítať nevieme. Napríklad nevieme vypočítať všetky dvojice relácie mensi(X, Y) ← X<Y (nepoznáme ani len typy X a Y).*

b) Vysvetlite niektorú metódu výpočtu bezpečných Datalogových programov. **(2)**

Univerzálnou metódou výpočtu Datalogových programov je naivná evaluácia. Nech program definuje predikáty p_1, ..., p_N v pravidlách r_1, ..., r_M. Každému predikátu priradíme reláciu P_i, i=1...N. Každé pravidlo preložíme do relačnej algebry (ako priradenie), čím dostaneme algebraické výrazy e_1, ..., e_M. Keď zanedbáme technické detaily, štruktúra každého výrazu e_i pre pravidlo tvaru
$p_j(vars) \leftarrow g_1, ..., g_q, \neg n_1, ..., \neg n_s.$
je

$P_j := (P_j \cup \pi_{vars}(G_1 \bowtie ... \bowtie G_q)) - N_1 - ... - N_s$

kde P_j, G_1, …, G_q, N_1, ..., N_s sú relácie priradené príslušným predikátom.
Teda najskôr sa preložia všetky pozitívne podciele do postupnosti joinov (technicky to je komplikovanejšie, kvôli premenovaniam atribútov resp. relácií, aritmetickým podcieľom, ...). Výsledok joinu sa zjednotí (po príslušnej projekcii) s reláciou P_j. Potom každý negovaný podcieľ sa preloží ako množinový rozdiel predošlého výrazu a relácie priradenej negovanému podcieľu. (Aj to je v skutočnosti technicky komplikovanejšie, lebo pred výpočtom rozdielu treba prispôsobiť atribúty výrazu na ľavej strane negovanému podcieľu; a po výpočte rozdielu treba znovu prijoinovať atribúty „stratené“ tým prispôsobovaním.)
Riešenie úlohy 1a ilustruje, ako sa tento strojový preklad urobí.

Algoritmus naivnej evaluácie:

```
/* initialisation */
for (i = 1 to N)
{
        Pi := {} /* empty set */
}
/* recursion */
do
{
        for (i = 1 to M)
        {
                compute expression ei for rule ri
        }
} while (some Pi, i=1...N, has changed inside this loop)
apply selection and projection determined by the query
```

Namiesto tej „cudzej“ syntaxe môžeme s použitím operátorov sekvencie priradení (;) a operátora pevného bodu (φ, fix-point) zapísať celú naivnú evaluáciu ako jeden výraz relačnej algebry:
$P_1 := \{\}; \ldots P_N := \{\}; \varphi(e_1; ...; e_M)$
V predošlom riadku sú „obsiahnuté“ (ak veríme Church-Turingovej téze) všetky algoritmické výpočty, podobne ako v univerzálnom Turingovom stroji. (Program je zakódovaný vo výrazoch e_1, ..., e_M, vstupom je extenzionálna databáza.)

c) Môže sa výpočet s použitím metódy z úlohy b) zacykliť? Odpoveď ÁNO resp. NIE zdôvodnite. (T.j. buď uveďte konkrétny príklad bezpečného Datalogového programu s dotazom a odsimulujte ho na konkrétnom naplnení extenzionálnej databázy až do momentu, v ktorom sa cyklí; alebo vysvetlite, prečo taký program neexistuje.) **(2)**

ÁNO, sú programy, pre ktoré sa výpočet naivnou evaluáciou zacyklí. (Platí to nielen pre naivnú evaluáciu. Keď programovací jazyk umožňuje vyjadriť cyklus resp. rekurziu, tak spravidla umožňuje vyjadriť aj nekonečný cyklus resp. rekurziu.)

Uvažujme napríklad program
r1: $p(X) \leftarrow r(X), \text{not } p(X)$.
s dotazom ?- p(X) a extenzionálnou databázou {r(X)={1}}.

Tento program je bezpečný, lebo jediná premenná X v jedinom pravidle r1 sa vyskytuje v pozitívnom relačnom podcieli r(X).

Pravidlo r1 preložené do relačnej algebry:
$P := (P \cup R) - P$

Algoritmus naivnej evaluácie pre daný program (predikátom r, p sú priradené relácie R, P):

```
/* initialisation */
P := {}

/* recursion */
do
{
        P := (P ∪ R) – P
} while (P has changed inside this loop)
```

0. iterácia (inicializácia):
$P = \varnothing$

1. iterácia do-while cyklu
$P = \{1\}$

2. iterácia do-while cyklu
$P = \varnothing$

3. iterácia do-while cyklu
$P = \{1\}$

...

Výpočet nikdy neskončí, lebo relácia P sa po každej iterácii zmení. Po párnych iteráciách do-while cyklu $P = \varnothing$, po nepárnych iteráciách $P = \{1\}$. (Tento program nemá pevný bod.)

3. Databázový server bol vypnutý. Pri opätovnom štarte obsahuje log-file databázového systému 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>.
Popíšte čo najpresnejšie algoritmus obnovy **(1)** a uveďte sekvenciu priradení, ktoré sa vykonajú počas obnovy v tomto konkrétnom prípade. **(2)**

Všeobecný algoritmus obnovy prechádza log-file najskôr zostupne (od konca log-file k začiatku). Počas tohto prechodu aktualizuje zoznamy redo_list a undo_list (undo_list aj redo_list sú na začiatku prázdne) a zároveň robí UNDO operácie pre transakcie z undo_list. Keď príde na začiatok log-file, začne vzostupný prechod, pri ktorom vykonáva REDO operácie pre transakcie z redo_list. Keď príde na koniec logfile, začne systém normálnu prevádzku.

Sekvencia priradení v tomto prípade:
/* Zostupný prechod */
B := 200 (undo T2)
A := 100 (undo T2)
/* Vzostupný prechod */
A := 100 (redo T0)
A := 70 (redo T1)

4. Relácia zoznam(Priezvisko, Meno, Cislo, ...) má 4 000 000 záznamov. Relácia je uložená v diskovom poli s blokmi veľkosti 4kB. V každom bloku je uložených 10 záznamov. Stredná doba prenosu diskového bloku (do RAM aj opačne) je 100 ms (=0,1 s). Pre vykonanie SQL dotazu
select z.Priezvisko, z.Meno, z.Cislo from zoznam z order by z.Priezvisko, z.Meno
je rezervovaných 500 blokov v RAM (bloky v RAM a na disku sú rovnako veľké).
a) Popíšte organizáciu pamäte a jednotlivé fázy (na úrovni vstupov a výstupov) fyzického operátora merge-sort pri vykonávaní daného dotazu. **(1)**

Nech M je počet blokov rezervovaných v RAM. V prvej fáze sa použije M blokov v RAM pre čítanie relácie r. Po utriedení záznamov v RAM sa všetky bloky RAM zapíšu na disk do behu (utriedeného súboru). Toto sa opakuje, kým sa nedočíta celá relácia r.

V druhej fáze sa behy zlučujú do jedného výsledného behu. M-1 blokov v RAM sa použije pre vstup behov (1 blok pre 1 beh), 1 blok sa použije ako výstupný. Výstupný blok sa po naplnení pripája k výslednému behu. Po zlúčení vstupných (krátkych) behov do jedného výsledného (dlhého) behu tie vstupné behy zaniknú.

Druhá fáza sa opakuje, až kým neostane jediný beh. Ten jediný beh obsahuje utriedenú reláciu r.

b) Odhadnite čas vykonania daného dotazu. Svoj odhad zdôvodnite. **(2)**

Relácia r je uložená v 400000 blokoch (= 4000000 / 10).

V prvej fáze sa všetky bloky prečítajú a rovnaký počet blokov sa zapíše do utriedených behov:
800 behov (= 400000 / 500) dĺžky 500 blokov, 800000 (= 2 * 400000) I/O operácií.

V druhej fáze sa postupne prečíta prvých 499 behov dĺžky 500 blokov, ktoré sa zlučujú do jedného behu. Prečíta sa 249500 blokov (= 499 * 500), rovnaký počet sa zapíše na disk:
301 behov dĺžky 500 blokov, 1 beh dĺžky 249500 blokov, 499000 I/O operácií (= 2 * 249500).

Druhá fáza sa zopakuje. Počet behov je už menší než 499, takže sa všetky podarí zlúčiť do jedného. Každý blok z tých 302 behov sa raz prečíta, rovnaký počet blokov sa zapíše na disk:
1 beh dĺžky 400000 blokov, 800000 (= 2 * 400000) I/O operácií.

Dokopy sa urobí 2099000 I/O operácií (= 4 * 400000 + 499000), čo trvá **209900 sekúnd**, čo je **zhruba 58 hodín**.

c) Ak sa namiesto 500 blokov rezervuje v RAM 5000 blokov (teda desaťkrát viacej), koľkokrát sa tým urýchli čas vykonávania daného dotazu? Zdôvodnite. **(2)**

V prvej fáze sa všetky bloky prečítajú a rovnaký počet blokov sa zapíše do utriedených behov:
400 behov (= 400000 / 1000) dĺžky 1000 blokov, 800000 (= 2 * 400000) I/O operácií.
V druhej fáze je počet behov menší než 1000, takže sa všetky podarí zlúčiť do jedného na jeden prechod. Každý blok každého behu sa raz prečíta, rovnaký počet blokov sa zapíše na disk:
1 beh dĺžky 400000 blokov, 800000 (= 2 * 400000) I/O operácií.

Dokopy sa urobí 1600000 I/O operácií (= 4 * 400000), čo trvá **160000 sekúnd**, čo je **zhruba 44 hodín**. To je zhruba **1,3-násobné zrýchlenie**.

Odhad 44 hodín sa nezmení ani pri rezervácii 800 či 8000 blokov RAM. Podstatné je, že druhú fázu vtedy netreba opakovať.
Pri horeuvedených odhadoch sme zanedbali zanedbali časy triedenia v RAM (aj preto, že s danými informáciami ich odhadnúť nevieme). To sa kompenzuje tým, že sme zanedbali tiež optimalizáciu prechodu medzi jednotlivými fázami merge-sort. Dá sa využiť, že niektoré bloky uložené v RAM sa dajú „recyklovať“, t.j. netreba ich opätovne čítať z disku.