18/1/2018 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 relačnom kalkule **(6)** a Datalogu **(6)** na dvojice
[P, K] také, že pijan P počas niektorej návštevy krčmy K vypil všetky svoje obľúbené alkoholy.

Relačný kalkul:

$$\{[P, K]: \exists I\ navstivil(I, P, K) \wedge \neg (\exists A\ lubi(P, A) \wedge \neg (\exists M\ vypil(I, A, M)))\}$$

Datalog:

```
answer(P, K) ←
        navstivil(I, P, K),
        not lubi_nevypil(P, I).

lubi_nevypil(P, I) ←
        navstivil(I, P, _),
        lubi(P, A),
        not v(I, A).

v(I, A) ←
        vypil(I, A, _).
```

b) Sformulujte bezpečný dotaz v Datalogu **(6)** a SQL **(6)** na krčmy K také, že viacej než polovica pijanov, ktorí K navštívili, v K nikdy nič nevypila.

Datalog:

```
answer(K) ←
        subtotal(navstevnici(P, K), [K], TN = count(P)),
        subtotal(pijani(P, K), [K], TP = count(P)),
        2 * TP < TN. /* TP2 is 2 * TP, TP2 < TN */

navstevnici(P, K) ←
        navstivil(_, P, K).

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

SQL:

```
with
navstevnici as (
        select n.Krcma, count(distinct n.Pijan) as TN
        from navstivil n
        group by n.Krcma
),

pijani as (
        select n.Krcma, count(distinct n.Pijan) as TP
        from navstivil n, vypil v
        where n.Idn = v.Idn
        group by n.Krcma
)

select n.Krcma
from navstevnici n, pijani p
where n.Krcma = p.Krcma and 2 * p.TP < n.TN
```

2. SQL príkaz *create assertion* definuje integritné obmedzenie na databázu. Systém dovolí aktualizáciu databázy len keď sa podmienka v klauze *check* vyhodnotí po aktualizácii ako *true* (vyhodnocuje sa rovnako ako podmienka v klauze *where* príkazu *select*); inak abortuje transakciu, ktorá tú aktualizáciu robí. Napríklad, integritné obmedzenie Mnozstvo > 0 pre databázu z úlohy 1 sa vyjadrí takto:
create assertion asrt_mn_positive check (
*not exists (select * from vypil v where v.Mnozstvo <= 0))*
a) Uvažujte databázu z úlohy 1. Vyjadrite funkčnú závislosť
Idn, Alkohol $\rightarrow$ *Mnozstvo* príkazom *create assertion*. **(6)**

```
create assertion asrt_fd_idn_a_m check (
        not exists (
                select *
                from vypil v1, vypil v2
                where v1.Idn = v2.Idn and v1.Alkohol = v2.Alkohol and v1.Mnozstvo <> v2.Mnozstvo
        )
)
```

b) Každá hodnota *Idn*, ktorá sa vyskytuje v relácii *vypil*, sa musí vyskytovať aj v relácii *navstivil*. Vyjadrite toto prirodzené integritné obmedzenie príkazom *create assertion*. **(6)**

```
create assertion asrt_foreign_key_idn (
        not exists (
                select *
                from vypil v
                where not exists (
                        select *
                        from navstivil n
                        where n.Idn = v.Idn
                )
        )
)
```

3. Uvažujte Datalogový program nad extenzionálnou databázou a(X), b(X, Y), c(X):
$p(X) \leftarrow a(X), \neg r(X).$
$r(X) \leftarrow b(X, Y), \neg r(Y).$
$r(X) \leftarrow c(X).$
a) Zapíšte výpočet dotazu ?– p(X) naivnou evaluáciou v relačnej algebre. **(6)**

Dôsledná naivná evaluácia:
p = {};
r = {};
φ(
    p = a – r;
    $r = \pi_X (b \triangleright \rho_{r(Y)} (r)) \cup c;$
)

Predikát r nie je závislý na predikáte p. Tým pádom je možné najskôr (iteratívne) vypočítať r a až následne p.
Optimalizovaná verzia:
r = {};
φ(
    $r = \pi_X (b \triangleright \rho_{r(Y)} (r)) \cup c;$
)
p = a – r;

b) Okomentujte postup výpočtu dotazu ?– p(X) naivnou evaluáciou pre databázu
a(X) = {1, 2, 3, 4}, b(X, Y) = {[1, 2], [2, 3]}, c(X) = {3}.
Rozhodnite či výpočet skončí. Ak nie, vysvetlite prečo. Ak áno, uveďte výsledok dotazu. **(6)**

Počítajme podľa optimalizovanej verzie (ušetríme niekoľko priradení do relácie p, lebo zaujímavé je len to posledné):

Po prvom priradení je r prázdna relácia:
r = {}

Po 1. iterácii φ obsahuje r všetky hodnoty b.X a c.X:
r = {1, 2, 3}

Po 2. iterácii φ obsahuje r všetky hodnoty b.X a c.X, okrem *1* a *2* (lebo b(*1*, 2) sa joinuje s r(2) a b(*2*, 3) sa joinuje s r(3)):
r = {3}

Po 3. iterácii φ obsahuje r všetky hodnoty b.X a c.X okrem *2* (lebo b(*2*, 3) sa joinuje s r(3)):
r = {1, 3}

Po 4. iterácii φ obsahuje r všetky hodnoty b.X a c.X okrem *2* (lebo b(*2*, 3) sa joinuje s r(3))::
r = {1, 3}

Relácia r sa nezmenila, operátor φ dosiahol pevný bod.

Po finálnom priradení do relácie p obsahuje p všetky hodnoty z a, ktoré nie sú v r:
p = {2, 4}

Výpočet dotazu ?– p(X) skončí, výsledkom je {2, 4}.

4. a) Dokážte, že scheduler používajúci dvojfázový zamykací protokol generuje len rozvrhy, ktoré sú konflikt-sériovateľné. **(6)**

Dôkaz (nepriamo). Uvažujme rozvrh, ktorý je generovaný 2PL schedulerom a ktorý nie je konflikt-sériovateľný. To znamená, že graf konfliktov transakcií pre tento (výstupný) rozvrh obsahuje nejaký cyklus $T_1 \rightarrow T_2 \rightarrow ... \rightarrow T_n \rightarrow T_1$. Vrcholy uzlov resp. identifikátory príslušných (navzájom rôznych) transakcií v tom cykle sme bez ujmy na všeobecnosti označili $T_1, ..., T_n$, kde $n > 1$. Každá z transakcií $T_1, ..., T_n$ je v tom rozvrhu commitovaná. Hrana $T_i \rightarrow T_j$ je v grafe konfliktov práve vtedy, keď niektorá operácia T_i je v tom rozvrhu skôr než niektorá konfliktná operácia od T_j.

Z 2PL vyplýva (v 2PL transakcia pri čítaní/zápise dátového objektu drží príslušný zámok; a kým ho drží, nemôže iná transakcia vykonať konfliktnú operáciu na tom dátovom objekte), že transakcia T_2 sa mohla dostať do konfliktu s T_1 až po tom, čo T_1 uvoľnila zámok na nejaký dátový objekt (ktorý predtým s prideleným zámkom čítala resp. zapisovala). Až po uvoľnení tohto zámku musela T_2 ten zámok získať, aby čítaním resp. zápisom do rovnakého dátového objektu spôsobila konflikt.

Tvrdenie z predošlého odstavca platí aj pre dvojicu transakcií T_n a T_1.

To však znamená, že T1 musela nejaký zámok (pre ten objekt, ktorý spôsobil konflikt $T_n \rightarrow T_1$) získať po tom, čo uvoľnila iný zámok (pre ten objekt, ktorý spôsobil konflikt $T_1 \rightarrow T_2$). Toto je v spore s dvojfázovosťou schedulera (lebo 2PL scheduler nepridelí zámok transakcii po tom, čo tá transakcia nejaký zámok predtým uvoľnila).

b) Vysvetlite metódu wait-die na prevenciu deadlockov pri dvojfázovom zamykaní. Uveďte príklad striktného a konflikt-sériovateľného vstupného rozvrhu (doplneného o operácie read-lock, write-lock, unlock), na ktorom metóda wait-die zbytočne abortuje niektorú transakciu. Okomentujte akcie schedulera po prečítaní každej vstupnej operácie. **(6)**

Wait-die metóda spočíva v nasledujúcich rozšíreniach 2PL schedulera:
Každá transakcia T dostane pri vykonaní $start_T$ časovú pečiatku TS(T).
Keď transakcia T žiada o zámok (t.j. scheduler vykonáva $read\text{-}lock_T$ alebo $write\text{-}lock_T$), ktorý nemôže byť ihneď pridelený lebo je v konflikte so zámkom prideleným inej transakcii T2, tak scheduler spustí nasledujúcu procedúru:

```
if (TS(T) < TS(T2))
        ... pozdrž vykonávanie T, kým sa príslušný zámok uvoľní;
else
        abort T;
```

Inak povedané, čakať na uvoľnenie zámku smie len transakcia, ktorá je staršia (začala skôr) než transakcia, ktorá ten zámok drží.

Príklad striktného a konflikt-sériovateľného vstupného rozvrhu, na ktorom wait-die zbytočne abortuje niektorú transakciu:
$start_1$, *$read\text{-}lock_1$ (X)*, $read_1$ (X), $start_2$, *$write\text{-}lock_2$ (X)*, $commit_1$, $write_2$ (X), $commit_2$
(Tento rozvrh je striktný, lebo neobsahuje dirty-read ani dirty-write. Je konflikt-sériovateľný, lebo je konflikt-ekvivalentný sériovému rozvrhu $T_1 \rightarrow T_2$.)
Pri čítaní tohto rozvrhu robí 2PL scheduler s wait-die toto:

$start_1$	pridelí TS(T1) transakcii T1
$read\text{-}lock_1$ (X)	pridelí read-lock na X transakcii T1
$read_1$ (X)	prečíta X pre transakciu T1
$start_2$	pridelí TS(T2) transakcii T2
$write\text{-}lock_2$ (X)	zistí, že nemôže prideliť $write\text{-}lock_2$(X) kvôli už pridelenému $read\text{-}lock_1$(X), porovná TS(T1) a TS(T2) a abortuje T2