3/1/2025 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), SQL (6) a relačnej algebre (6) na dvojice [P, K] také, že pijan P pri každej návšteve krčmy K vypil rovnakú množinu alkoholov; a zároveň P ľúbi všetky alkoholy, ktoré K čapuje.

Datalog:

```
answer(P, K) ←
        lubi(P, A),
        capuje(K, A),
        not capuje_nelubi(P, K),
        not rozne_mnoziny(P, K).

capuje_nelubi(P, K) ←
        lubi(P, _),
        capuje(K, A),
        not lubi(P, A).

rozne_mnoziny(P, K) ←
        navstivil(I, P, K),
        v(I, A),
        navstivil(I2, P, K),
        not v(I2, A).

v(I, A) ←
        vypil(I, A, _).
```

SQL:

```
with
capuje_nelubi as (
        select l.Pijan, c.Krcma
        from lubi l, capuje c
        where not exists (
                select *
                from lubi l2
                where l2.Pijan = l.Pijan and l2.Alkohol = c.Alkohol),
rozne_moziny as (
        select
        from navstivil n, vypil v, navstivil n2
        where n.Idn = v.Idn and  n.Pijan = n2.Pijan and n.Krcma = n2.Krcma and not exists (
                select *
                from vypil v2
                where v2.Idn = n2.Idn and v2.Alkohol = v.Alkohol)
select l.Pijan, c
from lubi l, capuje c
where l.Alkohol = c.Alkohol and not exists (
                select *
                from capuje_nelubi cn
                where cn.Pijan = l.Pijan and cn.Krcma = c.Krcma)
        and not exists (
                select *
                from rozne_mnoziny rm
                where rm.Pijan = l.Pijan and rm.Krcma = c.Krcma)
```

Relačná algebra:

capuje_nelubi := $\pi_{Pijan,\ Krcma}$ ((π_{Pijan} (lubi) × capuje) ▷ lubi)

rozne_mnoziny := $\pi_{Pijan,\ Krcma}$ ((navstivil ⋈ vypil $\bowtie_{n.Pijan = n2.Pijan \wedge n.Krcma = n2.Krcma}$ ρ_{n2} (navstivil))

▷$_{v2.Idn = n2 \wedge Idn \wedge v2.Alkohol = vypil.Alkohol}$ ρ_{v2} (vypil))

/* answer */

(lubi ⋈ capuje) ▷ capuje_nelubi ▷ rozne_mnoziny

b) Sformulujte bezpečný dotaz v Datalogu (6) na dvojice [P, A] také, že pijan P ľúbi alkohol A; a v každej krčme, ktorú P navštívil, vypil P alkohol A v menšom celkovom množstve než ktorýkoľvek iný návštevník tej krčmy.

```
answer(P, A) ←
        lubi(P, A),
        not iny_niekde_tolko(P, A).

iny_niekde_tolko(P, A) ←
        vypite(P, K, A, T),
        vypite(P2, K, A, T2),
        not P2 = P,
        T2 <= T.

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

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

vypite(P, K, A, 0) ←
        navstivil(_, P, K),
        not nv(P, K, A).

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

2. a) Napíšte v zrozumiteľnom pseudokóde algoritmus s polynomiálnou časovou zložitosťou, ktorý pre danú relačnú schému [r, F] nájde (vypíše) niektorý kľúč tej schémy. Ak taký algoritmus neexistuje, vysvetlite prečo; inak zdôvodnite korektnosť a časovú zložitosť svojho algoritmu. **(6)**

Taký algoritmus existuje.

Na vstupe je relácia $r(A_1, ..., A_N)$ a množina funkčných závislostí F. Ideou je redukcia ľavej strany platnej funkčnej závislosti $A_1, ..., A_N \rightarrow A_1, ..., A_N$. Ak sa dá odstrániť niektorý z atribút z ľavej strany, bez porušenia platnosti tej redukovanej závislosti, tak sa odstráni. Keď už ľavá strana neobsahuje redundantný atribút, tak je z definície kľúčom r.

```
K = {A1, ..., AN};
for (i = 1; i <= N; i++)
{
        if ((K – {Ai})* = {A1, ..., AN})
                K = K – {Ai};
}
print(K);
```

Tento algoritmus má polynomiálnu časovú zložitosť, lebo výpočet uzáver množiny atribútov sa dá vypočítať v polynomiálnom čase, a ten uzáver sa počíta len N-krát.

b) Napíšte v zrozumiteľnom pseudokóde algoritmus s polynomiálnou časovou zložitosťou, ktorý rozhodne (vypíše ÁNO resp. NIE), či daná relačná schéma [r, F] má aspoň dva rôzne kľúče. Ak taký algoritmus neexistuje, vysvetlite prečo; inak zdôvodnite korektnosť a časovú zložitosť svojho algoritmu. **(6)**

Taký algoritmus existuje.
Na vstupe je relácia $r(A_1, ..., A_N)$ a množina funkčných závislostí F. Využijeme algoritmus z úlohy a), ktorý nájde nejaký kľúč r. Nech tento kľúč pozostáva z atribútov $B_1, ..., B_M$. Ak existuje aj nejaký iný kľúč, tak obsahuje len nejakú striktnú podmnožinu atribútov $B_1, ..., B_M$. Stačí testovať, či $A_1, ..., A_N$ je aj po odobratí niektorého atribútu B_i nadkľúčom r, t.j. či uzáver tejto množiny obsahuje odobratý atribút B_i.
(Ak $\{A_1, ..., A_N\} – \{B_i\}$ je nadkľúč r, tak ten iný kľúč vieme nájsť minimalizáciou ľavej strany funkčnej závislosti $\{A_1, ..., A_N\} – \{B_i\} \rightarrow A_1, ..., A_N$, podobne ako v úlohe a)).

```
K = {B1, ..., BM}; /* kľúč r */
found = FALSE;
for (i = 1; i <= M; i++)
{
        if (Bi ∈ ({A1, ..., AN} – {Bi})*)
                found = TRUE;
}
if (found)
        print(„ÁNO“);
else
        print(„NIE“);
```

Tento algoritmus má polynomiálnu časovú zložitosť, lebo výpočet uzáver množiny atribútov sa dá vypočítať v polynomiálnom čase, a ten uzáver sa počíta len N-krát.

3. a) Uvažujte relácie r(X, Y, U), s(X, Y, V) (bez duplikátov a null hodnôt) a SQL dotaz
select distinct r.X, r.U from r
*where not exists (select * from s where s.X = r.X or s.Y = r.Y).*
Zapíšte tento dotaz ekvivalentne v SQL, bez použitia *or* v klauze *where*. **(6)**

select distinct r.X, r.U from r
where not exists (select * from s where s.X = r.X) and not exists (select * from s where s.Y = r.Y)

b) Rozhodnite, či je možné ľubovoľný bezpečný Datalogový program s dotazom vyjadriť ekvivalentne v relačnom kalkule. Odpoveď ÁNO resp. NIE zdôvodnite; t.j. buď vysvetlite ako sa to urobí, alebo vysvetlite prečo sa to nie vždy dá. **(6)**

ÁNO, lebo Datalog je syntaktickým zúžením relačného kalkulu.

Datalogový program P s dotazom Q (dotaz Q je atomická formula, t.j. predikát s premennými) sa do relačného kalkulu prepíše ako
{var(Q): Q ∧ Φ}, kde var(Q) je usporiadaná n-tica premenných v Q.
Formula Φ je konjunkcia implikácií, ktoré zodpovedajú pravidlám v P. V každej implikácii sú všetky premenné kvantifikované univerzálne (globálne, pre celú implikáciu).
Preklad jednotlivého Datalogového pravidla vyžaduje ešte dodatočné textové zámeny:
’,’ sa preloží do ’∧’, ’not’ sa preloží do ’¬’, a podobne.

4.
a) Rozhodnite či základná verzia metódy časových pečiatok pre izoláciu transakcií garantuje aj obnoviteľnosť výstupného rozvrhu. Odpoveď ÁNO resp. NIE zdôvodnite. **(6)**

NIE.
Uvažujme tento vstupný rozvrh, ktorý nie je obnoviteľný (kvôli dirty-read a poradiu commitov):
s1, s2, w1(X), r2(X), c1, c2
Scheduler s metódou časových pečiatok vykoná všetky tieto operácie v tom poradí, ako prichádzajú:
Pri operácii w1(X) vyhodnotí $TS(1) < TSR(X) \wedge TS(1) < TSW(X)$ ako TRUE, takže ten zápis vykoná a TSW(X) aktualizuje na TS(1). Pri r2(X) vyhodnotí $TS(2) < TSW(X)$ ako TRUE, takže to čítanie vykoná a TSR(X) aktualizuje na TS(2). Operácie commit vykoná.
Tým pádom výstupný rozvrh nie je obnoviteľný.

b) Databázový systém vypadol. Pred opätovným spustením obsahuje log-file nasledujúce záznamy: <T1, start>, <T1, X, 1, 2>, <T2 start>, <T1, Y, 2, 3>, <T2, Z, 0, 1>, <T1, commit>, <T3, start>, <T3, X, 2, 4>. Popíšte priebeh všeobecného algoritmu obnovy v tomto konkrétnom prípade, a uveďte sekvenciu priradení, ktoré sa vykonajú počas obnovy. **(6)**

Pri obnove sa log číta najskôr od konca, vytvára sa undo-list a redo-list, a aplikujú sa UNDO operácie pre transakcie v undo-list.
Priebeh UNDO prechodu:

Log záznam	Akcia
	undo_list = {}; redo_list = {};
<T3, X, 2, 4>	undo_list = {T3}; X = 2;
<T3, start>	undo_list = {};
<T1, commit>	redo_list = {T1};
<T2, Z, 0, 1>	undo_list = {T2}; Z = 0;
<T1, Y, 2, 3>	
<T2 start>	undo_list = {};
<T1, X, 1, 2>	
<T1, start>	
<BOF>	

Nasleduje REDO prechod, počas ktorého sa log číta od začiatku, a aplikujú sa REDO operácie pre transakcie v redo-list.
Priebeh REDO prechodu:

<T1, start>	
<T1, X, 1, 2>	X = 2;
<T2 start>	
<T1, Y, 2, 3>	Y = 3;
<T2, Z, 0, 1>	
<T1, commit>	
<T3, start>	redo_list = {};

Tu algoritmus obnovy končí, lebo redo_list je prázdny.

Sekvencia priradení (zápisov do databázy) počas obnovy: X = 2; Z = 0; X = 2; Y = 3.