21/1/2026 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)** a SQL **(6)** na dvojice [P, A] také, že pijan P niekedy vypil alkohol A; a alkohol A sa čapuje v každej takej krčme, v ktorej P vypil všetky svoje obľúbené alkoholy počas niektorej jednej návštevy.

Datalog:

```
answer(P, A) ←
        navstivil(I, P, _),
        vypil(I, A, _),
        not splnil_necapuje(P, A).

splnil_necapuje(P, A) ←
        navstivil(I, P, K),
        vypil(_, A, _),
        not nevypil_nieco_oblubene(I),
        not capuje(K, A).

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

v(I, A) ←
        vypil(I, A, _).
```

SQL:

```
with
nevypil_nieco oblubene as
(
select n.Idn
from navstivil n, lubi l
where n.Pijan = l.Pijan and not exists (
        select *
        from vypil v
        where v.Idn = n.Idn and v.Alkohol = lubi.Alkohol)
),
splnil_necapuje as
(
select n.Idn, n.Pijan
from navstivil n, vypil v
where not exists (
        select *
        from nevypil_nieco_oblubene nno
        where n.Idn = nno.Idn)
        and not exists (
        select *
        from capuje c
        where c.Krcma = n.Krcma and c.Alkohol = v.Alkohol)
)
select n.Pijan, v.Alkohol
from navstivil n, vypil v
where n.Idn = v.Idn and not exists (
        select *
        from splnil_necapuje sn
        where sn.Pijan = n.Pijan and sn.Alkohol = v.Alkohol)
```

b) Sformulujte bezpečný dotaz v Datalogu **(6)** a SQL **(6)** na dvojice [K, A] také, že alkohol A vypila v K viac než polovica (rôznych) návštevníkov krčmy K; a zároveň A sa vypil v K v menšom celkovom množstve než ľubovoľný iný alkohol, ktorý sa v K čapuje.

Datalog:

```
answer(K, A) ←
        subtotal(np(K, A, P), [K, A], [C = count(P)]),
        subtotal(n(K, P), [K], [T = count(P]),
        2 * C > T,
        not ineho_tolko(K, A).

ineho_tolko(K, A) ←
        subtotal(nv(_, K, A, M), [K, A], [S = sum(M)]).
        subtotal(nv(_, K, A2, M2), [K, A2], [S2 = sum(M2)]).
        not A = A2,
        S2 >= S1.

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

n(K, P) ←
        navstivil(_, P, K).

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

SQL:

```
with
subtotal_np as
(
select n.Krcma, v.Alkohol, count(distinct n.Pijan) as C
from navstivil n, vypil v
where n.Idn = v.Idn
group by n.Krcma, v.Alkohol
),
subtotal_n as
(
select n.Krcma, count(distinct n.Pijan) as T
from navstivil n
),
subtotal_nv as
(
select n.Krcma, v.Alkohol, sum(v.Mnozstvo) as S
from navstivil n, vypil v
where n.Idn = v.Idn
group by n.Krcma, v.Alkohol
),
ineho_tolko as
(
select snv1.Krcma, snv1.Alkohol
from subtotal_nv snv1, subtotal_nv snv2
where snv1.Krcma = snv2.Krcma and snv1.Alkohol <> snv2.Alkohol and snv2.S >= snv1.S
)
select snp.Krcma, snp.Alkohol
from subtotal_np snp, subtotal_n as sn
where snp.Krcma = sn.Krcma and snp.Pijan = sn.Pijan and 2 * C > T and not exists (
        select *
        from ineho_tolko it
        where it.Krcma = snp.Krcma and it.Alkohol = snp.Alkohol
)
```

2. Uvažujte relácie r(A, B), s(C) (bez duplikátov a bez NULL hodnôt).
a) Napíšte nasledujúci SQL dotaz ekvivalentne v Datalogu: **(6)**

```
select distinct s1.C
from s s1
where not exists
        (select * from s s2, r r1, r r2 where r1.B = s1.C or r1.A = r2.B)
```

```
answer(C) ←
        s(C),
        not inner(C).

inner(C) ←
        s(_),
        r(_, C),
        r(_, _).

inner(C) ←
        s(C), /* safety */
        s(_),
        r(A, _),
        r(_, A).

?- answer(C).
```

b) Vypočítajte výsledok dotazu z úlohy a) pre reláciu
r(A, B) = {[3, 6], [4, 8], [5, 2], [5, 6], [8, 4], [9, 6]}, s(C) = {1, 3, 5, 7, 9}. **(6)**

Všimnime si to druhé pravidlo pre *inner*. Neformálne hovorí toto (uvažujeme o reláciách, ktoré prislúchajú predikátom pri naivnej evaluácii tohto Datalogového programu): ak *s* je neprázdna relácia, a ak existuje hodnota *A*, ktorá je v *r* na prvom mieste niektorej dvojice a tiež na poslednom mieste niektorej dvojice, tak relácia *inner* je rovná relácii *s*. (To prvé pravidlo by mohlo do *inner* len pridať ďalšie záznamy, ale tie nás nemusia zaujímať.) Všetko tie predpoklady sú splnené, lebo *s* je neprázdna relácia, a *r* obsahuje záznamy [4, 8] a [8, 4]. Takže $inner \supseteq s$.

Relácia *answer* sa počíta ako $s - inner$. Tým pádom **výsledok dotazu je prázdna množina**.

3. Dané sú relácie *r(X, Y, Z)* a *s(X)* (bez duplikátov a null hodnôt).
a) Definujte v Datalogu reláciu *t*, pre ktorú platí $s \times t = r$. (Relácie interpretujte v Datalogu ako predikáty.) **(6)**

Relácia sa v Datalogu interpretuje rovnomenným charakteristickým predikátom, ktorého počet argumentov je rovný počtu atribútov tej relácie, a ktorý platí práve pre tie n-tice, ktoré patria do tej relácie.

Kartézsky súčin $s \times t$ je relácia, ktorá je definovaná týmito vlastnosťami:
1. Jej atribúty sú zjednotením atribútov s a t.
2. Pre každú n-ticu $[s_1, ..., s_M]$ a pre každú n-ticu $[t_1, ..., t_N]$ platí, že ak $[s_1, ..., s_M] \in s$ a $[t_1, ..., t_N] \in t$, tak $[s_1, ..., s_M, t_1, ..., t_N] \in s \times t$.

Definícia predikátu cartesian(., ., .), ktorý interpretuje kartézsky súčin $s \times t$:

```
cartesian(X, Y, Z) ←
        s(X),
        t(Y, Z).
```

Definícia predikátu *t*:

```
t(Y, Z) ← /* t musí mať správne argumenty, aby bola splnená vlastnosť 1 */
        r(_, Y, Z), /* hodnoty Y a Z musia byť z r, inak by v s platilo aj to, čo nemá (nesmie) platiť */
        not nieco_chyba(Y, Z). /* žiadna kombinácia nesmie chýbať v r, aby bola splnená vlastnosť 2 */

nieco_chyba(Y, Z) ←
        r(_, Y, Z),
        s(X),
        not r(X, Y, Z).
```

*Nejaké podobnosti možno nájsť v celočíselnej aritmetike, bežne používanej v programovacích jazykoch. Napríklad, asi málokoho prekvapuje, že 7 / 3 = 2. Lenže ak platí 6 / 3 = 2, 7 / 3 = 2, 8 / 3 = 2, tak hádam aj opačne: 2 * 3 = 6, 2 * 3 = 7, 2 * 3 = 8.*

b) Zapíšte výpočet relácie *t* z úlohy a) v relačnej algebre. Vypočítajte reláciu *t* pre
r(X, Y, Z) = {[0, 0, 0], [0, 0, 1], [0, 1, 0], [1, 0, 0], [1, 0, 1], [1, 1, 1]},
s(X) = {[0], [1]}. **(6)**

Výpočet *t*:

nieco_chyba := $\pi_{Y, Z}((\pi_{Y, Z}(r) \times s) - r)$;

t := $\pi_{Y, Z}(r) - $ nieco_chyba;

Simulácia výpočtu po krokoch pre dané *r* a *s*:
nieco_chyba = {[1, 0], [1, 1]}
t = {[0, 0], [0, 1]}

4. Uvažujte transakčný systém, ktorý používa dvojfázové zamykanie. Do systému vstupujú nasledujúce dve transakcie (len tieto dve sú aktívne):
T1: s1, r1(X), r1(Y), w1(X), c1. T2: s2, r2(X), r2(Y), w2(Y), c2.
a) Rozšírte T1 a T2 o operácie rlock (read-lock), wlock (write-lock) a unlock tak, aby boli konformné s dvojfázovým zamykacím protokolom; dbajte na to, aby zámky boli získavané čím neskôr a uvoľňované čím skôr. Rozhodnite, či existuje rozvrh takto rozšírených T1 a T2, ktorý vedie k deadlocku T1 a T2. Odpoveď ÁNO resp. NIE zdôvodnite. **(6)**

Rozšírenie transakcií o získavanie a uvoľňovanie zámkov :
T1: start, wl1(X), r1(X), rl1(Y), r1(Y), ul1(Y), w1(X), ul1(X), commit.
T2: start, rl2(X), r2(X), wl2(Y), ul2(X), r2(Y), w2(Y), ul2(Y), commit.

Pri vykonávaní T1 a T2 sú dve možnosti:
1. wl1(X) sa vykoná pred rl2(X). Vtedy bude výstupný rozvrh sériový T1→T2, až na poradie vykonávania operácií start a commit. Deadlock vzniknúť nemôže.
2. rl2(X) sa vykoná pred wl1(X). Vtedy sa vynútene vykoná časť T2 po ul2(X). Výstupný rozvrh sériový T2→T1, až na poradie vykonávania operácií start a commit. Deadlock vzniknúť nemôže.

NIE. Pri vykonávaní týchto dvoch transakcií deadlock nevznikne.

b) Vysvetlite, ako funguje metóda wait-or-die na prevenciu deadlocku. Rozhodnite, či existuje rozvrh rozšírených T1 a T2, ktorý vedie k abortu niektorej transakcie, ak systém používa metódu wait-or-die. Odpoveď ÁNO resp. NIE zdôvodnite. **(6)**

Systém pridelí každej transakcii časovú pečiatku v čase vykonávania jej operácie start.
Ak nejaká transakcia T žiada o konfliktný zámok, ktorý v tom momente drží transakcia T', tak systém najskôr zistí, ktorá z nich je staršia (t.j. má menšiu časovú pečiatku). Ak je T staršia než T', tak systém nechá T čakať na pridelenie zámku. Inak systém abortuje T.

ÁNO. Po prečítaní rozvrhu
start2, rl2(X), start1, wl1(X)
bude T1 abortovaná.