3/1/2024 Úvod do databáz, skúškovýx 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 také trojice [K, A1, A2], že krčma K čapuje alkoholy A1 aj A2; a zároveň A1 a A2 sa pri každej návšteve krčmy K vypili len súčasne (t.j. pri každej návšteve K sa buď vypili A1 aj A2, alebo žiaden z nich).

Datalog:

```
answer(K, A1, A2) ←
        capuje(K, A1),
        capuje(K, A2),
        not vypity_nevypity(K, A1, A2),
        not vypity_nevypity(K, A2, A1).

vypity_nevypity(K, A1, A2) ←
        navstivil(I, _, K),
        vypil(I, A1, _),
        capuje(K, A2),
        not v(I, A2).

v(I, A) ←
        vypil(I, A, _).
```

SQL:

```
with vypity_nevypity as (
        select n.Krcma, v.Alkohol as A1, c.Alkohol as A2
        from navstivil n, vypil v, capuje c
        where n.Idn = v.Idn and n.Krcma = c.Krcma and not exists (
                select *
                from vypil v2
                where n.Idn = v2.Idn and c.Alkohol = v2.Alkohol
                )
)
select c1.Krcma, c1.Alkohol, c2.Alkohol
from capuje c1, capuje c2
where c1.Krcma = c2.Krcma and not exists (
        select *
        from vypity_nevypity vn
        where c1.Krcma = vn.Krcma and ((c1.Alkohol = vn.A1 and c2.Alkohol = vn.A2) or
        (c1.Alkohol = vn.A2 and c2.Alkohol = vn.A1))
        )
```

Relačná algebra:

$$\text{vypity\_nevypity} = \pi_{\text{Krcma, A1, A2}} (((\text{navstivil} \bowtie \rho_{\text{v(Idn, NA1, M)}} (\text{vypil})) \bowtie \rho_{\text{c(Krcma, NA2)}} (\text{capuje})) \triangleright \rho_{\text{v2(Idn, NA2, M2)}} (\text{vypil}))$$

/* answer */

$$(\rho_{\text{c(K, A1)}} (\text{capuje}) \bowtie \rho_{\text{c(K, A2)}} (\text{capuje})) \triangleright_{\text{K=Krcma} \wedge ((\text{A1 = NA1} \wedge \text{A2 = NA2}) \vee (\text{A1 = NA2} \wedge \text{A2 = NA1}))} \text{vypity\_nevypity}$$

b) Sformulujte bezpečný dotaz v Datalogu **(6)** na dvojice [K, P] také, že pijan P je v krčme K jednoznačným rekordérom v pití aspoň troch druhov alkoholu (v zmysle celkových množstiev alkoholov vypitých v K).

```
answer(K, P) ←
        vypite(K, P, A1, S1),
        vypite(K, P, A2, S2),
        vypite(K, P, A3, S3),
        not A1 = A2,
        not A1 = A3,
        not A2 = A3,
        not iny_rekorder(K, P, A1, S1),
        not iny_rekorder(K, P, A2, S2),
        not iny_rekorder(K, P, A3, S3).

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

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

iny_rekorder(K, P, A, S) ←
        vypite(K, P, A, S),
        vypite(K, P2, A, S2),
        not P2 = P,
        S2 >= S.
```

2.
a) Definujte pojem zachovania funkčných závislostí pri dekompozícii relačnej schémy. Vysvetlite prečo je výhodné funkčné závislosti pri dekompozícii zachovať. **(6)**

Definícia: Dekompozícia (r_1, F_1), ..., (r_n, F_n) zachováva funkčné závislosti relačnej schémy (r, F), ak $F^+ = (\cup F_i)^+$.

Ak dekompozícia zachováva funkčné závislosti, tak ku každej relácii dekompozície je možné definovať lokálne integritné obmedzenie (constraint, v SQL napr. už pri *create table*), ktoré prislúcha funkčným závislostiam zachovaným v tej relácii.
Ak niektorá funkčná závislosť zachovaná nie je, tak príslušné integritné obmedzenie treba definovať globálne pre celú databázu, čo znamená väčšiu réžiu pri aktualizácii dát.

b) Uvažujte relačnú schému r(A, B, C, D, E) s funkčnými závislosťami
A→CE, AD→B, BC→D, BE→A.
Rozhodnite, či existuje dekompozícia r do dvoch relácií, ktorá je v tretej normálnej forme a zároveň je bezstratová (t.j. spája sa bezstratovo). Odpoveď ÁNO resp. NIE zdôvodnite—t.j. buď uveďte takú dekompozíciu a vysvetlite prečo má všetky požadované vlastnosti; alebo vysvetlite, prečo taká dekompozícia neexistuje. **(6)**

Nájdime všetky kľúče r.
ABCDE
- -A: BCDE
  - -B: CDE
  - +B: BCDE
    - -C: BDE
      - -D: **BE**
      - +D: BD
    - +C: BC
- +A: ABD
  - -B: **AD**
  - +B: **AB**

Kľúče v r sú AB, AD a BE, iné nie sú.

Definícia: r je v 3NF, ak pre každú netriviálnu kánonickú funkčnú závislosť **X**→Y platnú v r platí, že buď **X** je nadkľúč r alebo Y je časťou nejakého kľúča.

r nie je v 3NF, lebo platí A→C, pričom A nie je nadkľúč r a C nepatrí do žiadneho kľúča.

Keďže A, B, D, E sú kľúčové atribúty, relácia r1={A, B, D, E} je v 3NF. Platia v nej funkčné závislosti A→E, AD→B, BE→A.

Atribút C musí byť v r2. Ak sa r1 a r2 majú spojiť bezstratovo, ich spoločné atribúty musia byť nadkľúčom v r1 alebo v r2. Pridajme do r2 atribút A. Relácia r2 = {A, C} je v 3NF, a spoločný atribút A je (nad)kľúčom v r2.

ÁNO. Dekompozíca **r1={A, B, D, E}, r2 = {A, C}** má všetky požadované vlastnosti. (Ďalšie také dekompozície sú r1={A, B, D, E}, r2 = {B, C, E}; r1={A, B, D}, r2 = {A, C, E}.)

3.
a) Napíšte nasledujúci Datalogový program s dotazom ekvivalentne v relačnom kalkule. **(6)**
p(X, Y) ← e(X, Y).
p(X, Y) ← p(X, Z), p(Z, Y).
q(X, Y, Z) ← p(X, Z), p(Z, Z), p(Z, Y).
? q(X, Y, 2).

$$\{[X, Y]: q(X, Y, 2) \wedge$$
$$(\forall X \forall Y\ e(X, Y) \Rightarrow p(X, Y)) \wedge$$
$$(\forall X \forall Y \forall Z\ (p(X, Z) \wedge p(Z, Y)) \Rightarrow p(X, Y)) \wedge$$
$$(\forall X \forall Y \forall Z\ (p(X, Z) \wedge p(Z, Z) \wedge p(Z, Y)) \Rightarrow q(X, Y, Z))$$
$$\}$$

b) Rozhodnite či je možné každý bezpečný dotaz v relačnom kalkule vyjadriť ekvivalentne ako bezpečný Datalogový program (s dotazom). Odpoveď ÁNO resp. NIE zdôvodnite. **(6)**

ÁNO. Datalog bol v tomto kurze prezentovaný ako syntaktické zúženie relačného kalkulu, ktoré neznižuje vyjadrovaciu silu pôvodného jazyka.

Túto otázku je možné interpretovať aj inak: existuje algoritmus, ktorý ľubovoľnú bezpečnú formulu relačného kalkulu prepíše do ekvivalentného bezpečného programu v Datalogu? Pre nejakú triedu formúl je ten prepis priamočiary, všeobecne môže byť ťažký. Jazyk relačného kalkulu je príliš bohatý v zmysle, že umožňuje formulovať aj „nerozumné" dotazy. Napríklad, výsledkom nasledujúceho (bezpečného) dotazu

$$\{Z: q(Z) \wedge$$
$$(\forall X\ e(X) \Rightarrow (p(X) \vee r(X))) \wedge$$
$$(\forall X\ \ (p(X) \vee r(X)) \Rightarrow q(X))$$
$$\}$$

kde e(.) je konečná extenzionálna databáza, je zrejme množina všetkých Z, pre ktoré platí e(Z). Toto sa v Datalogu vyjadriť dá.
Ale nevedno, čo obsahuje táto množina:

$$\{Z: p(Z) \wedge$$
$$(\forall X\ e(X) \Rightarrow (p(X) \vee r(X))) \wedge$$
$$(\forall X\ \ (p(X) \vee r(X)) \Rightarrow q(X))$$
$$\}$$

Pokiaľ výsledky niektorých bezpečných dotazov v relačnom kalkule nie sú definované, a pokiaľ nie je známe ako popísať (celú) triedu dotazov s nedefinovaným výsledkom, nemá zmysel uvažovať o ich ekvivalentnom algoritmickom prepise.

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

NIE. Napríklad pre vstupný rozvrh
s1, s2, w1(X), r2(X), w2(Y), c2, c1
sa všetky operácie vykonajú v tomto poradí, t.j. výstupný rozvrh je identický so vstupným. Ale tento rozvrh nie je obnoviteľný, lebo r2(X) je dirty read (transakcia 2 číta necommitovanú hodnotu od transakcie 1) a c2 sa vykonal pred c1.

b) V základnej verzii metódy časových pečiatok sa pre vstupnú operáciu $w_T(X)$ robí toto:
if ((TS(T) < TSR(X)) || (TS(T) < TSW(X))) abort T;
else {TSW(X) = TS(T); execute($w_T(X)$);}
Nahraďme tento fragment kódu nasledujúcim:
if (TS(T) < TSR(X)) abort T;
else if (TS(T) > TSW(X)) {TSW(X) = TS(T); execute($w_T(X)$);}
/* inak neurob nič */
Rozhodnite či aj takto upravený scheduler garantuje (view-) sériovateľnosť výstupného rozvrhu. Odpoveď ÁNO resp. NIE zdôvodnite. **(6)**

Metóda časových pečiatok sériuje transakcie v poradí svojich časových pečiatok. T.j. výstupný rozvrh je (prinajmenšom) view-ekvivalentný sériovému rozvrhu, v ktorom sú commitované transakcie vykonané vcelku v poradí, v ktorom prídu do schedulera ich operácie start.

Jediná situácia, v ktorej sa ten pôvodný a modifikovaný kód správajú rôzne, je
$(TS(T) > TSR(X)) \wedge (TS(T) < TSW(X))$.
V tejto situácii pôvodný kód abortuje T, zatiaľ čo modifikovaný kód neurobí nič.
Prvá časť tej podmienky, TS(T) > TSR(X), hovorí, že $w_T(X)$ nie je v read-write konflikte na objekte X s read operáciami starších transakcií. Tým pádom vykonanie $w_T(X)$ neovplyvňuje sériovateľnosť výstupného rozvrhu.
Druhá časť tej podmienky, TS(T) < TSW(X), hovorí, že pred uvažovanou operáciou $w_T(X)$ bol už vykonaný write od niektorej mladšej transakcie. Ak sa v tomto prípade neurobí nič, tak to neohrozí view-sériovateľnosť. Pre view-sériovateľnosť výstupného rozvrhu je dôležitý len posledný zápis do objektu X, a ten je od tej mladšej transakcie (ktorá je sériovaná neskôr ako T).

Takto modifikovaná metóda časových pečiatok v niektorých prípadoch vedie ku commitu transakcie, ktorú základná metódu zbytočne abortuje . Pozornosť však treba venovať prípadu, keď tá mladšia transakcia z predošlého odstavca neskôr skončí abortom. Vtedy treba abortovať tiež transakciu T, inak by vo výslednom rozvrhu chýbala jej operácia $w_T(X)$. (Implementácia operácie undo je v kompetencii recovery managera; scheduler musí len zabezpečiť, aby recovery manager vedel tento prípad detekovať.)