30/1/2014 Úvod do databáz, **skúškový** test, max 25 bodov, 90 min

1. Daná je databáza (bez duplikátov a null hodnôt): capuje(Krcma, Alkohol), navstivil(Idn, Pijan, Krcma), vypil(Idn, Alkohol, Mnozstvo).

Platí: Idn → Pijan, Krcma; Krcma, Idn, Alkohol → Mnozstvo; Mnozstvo > 0.

a) Sformulujte nasledujúci dotaz v relačnom kalkule, **(2)**, Datalogu **(2),** SQL **(2)** a relačnej algebre **(2)**. Nájdite dvojice [A, K] také, že alkohol A čapovaný v krčme K vypil (pri aspoň jednej návšteve) každý pijan, ktorý K niekedy navštívil.

Relačný kalkul:

```
{[A, K]:
        capuje(K, A) ∧ ¬ (
                /* navstivil_nevypil(K, A) */
                ∃I ∃P
                navstivil(I, P, K) ∧ ¬ (
                        /* niekedy_vypil(P, K, A) */
                        ∃I2 ∃M
                        navstivil(I2, P, K) ∧ vypil(I2, A, M)
                )
        )
}
```

Datalog:

```
answer(A, K) ←
        capuje(K, A),
        not navstivil_nevypil(K, A).

navstivil_nevypil(K, A) ←
        capuje(K, A),  /* safety */
        navstivil(_, P, K),
        not niekedy_vypil(P, K, A).

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

SQL:

```
create temporary table niekedy_vypil as
select n.Pijan, n.Krcma, v.Alkohol
from navstivil n, vypil v
where n.Idn = v.Idn

create temporary table navstivil_nevypil as
select
from capuje c, navstivil n
where c.Krcma = n.Krcma and not exists (
        select *
        from niekedy_vypil nv
        where nv.Pijan = n.Pijan and nv.Krcma = c.Krcma and nv.Alkohol = c.Alkohol)

/* answer */
select c.Alkohol, c.Krcma
from capuje c
where not exists (
        select *
        from navstivil_nevypil nn
        where nn.Krcma = c.Krcma and nn.Alkohol = c.Alkohol)
```

Relačná algebra:

niekedy_vypil := $\pi_{Pijan,\ Krcma,\ Alkohol}$ (navstivil $\bowtie$ vypil)

navstivil_nevypil := $\pi_{Krcma,\ Alkohol}$ (

$\pi_{Pijan,\ Krcma,\ Alkohol}$ (capuje $\bowtie$ navstivil) – niekedy_vypil)

answer := capuje – navstivil_nevypil

b) Sformulujte nasledujúci dotaz v Datalogu **(2)** a SQL **(2)**. Nájdite dvojice [P, K] také, že v krčme K pijan P vypil viacej druhov alkoholov než ktorýkoľvek iný pijan.

Datalog:

```
answer(P, K) ←
        navstivil(_, P, K),
        not iny_majster(P, K).

iny_majster(P, K) ←
        subtotal(data(P, K, A), [P, K], [C = count(A)]).
        subtotal(data(P2, K, A), [P2, K], [C2 = count(A)]).
        C2 >= C,
        not P = P2.

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

SQL:

```
create temporary table pocet_alkoholov as
select n.Pijan, n.Krcma, count(distinct Alkohol) as C
from navstivil n, vypil v
where n.Idn = v.Idn
group by n.Pijan, n.Krcma

/* answer */
select pa1.Pijan, pa1.Krcma
from pocet_alkoholov pa1
where not exists (
        select *
        from pocet_alkoholov pa2
        where pa1.Krcma = pa2.Krcma and pa1.Pijan <> pa2.Pijan and pa2.C >= pa1.C)
```

2. Daná je relácia r(A, B, C, D, E, F, G) s funkčnými závislosťami
AB→C, ACEF→G, BC→AD, D→E, DE→F, DF→C, F→DE, G→F.
a) Nájdite všetky kľúče relácie r. **(1)**

Atribút B je v každom kľúči, lebo nie je na pravej strane žiadnej funkčnej závislosti. Všetky kľúče nájdeme úplným prehľadávaním podmnožín {A, B, C, D, E, F, G}.

ABCDEFG
- -A: BCDEFG
  - -C: BDEFG
    - -D: BEFG
      - -E: BFG
        - -F: **BG**
        - +F: **BF**
      - +E: BE
    - +D: **BD**
  - +C: **BC**
- +A: **AB**

Kľúčmi v r sú AB, BC, BD, BF, BG. Iné kľúče nie sú.

b) Nájdite minimálne pokrytie funkčných závislostí. **(1)**

Po minimalizácii ľavých strán kánonických funkčných závislostí:
AB→C, AF→G, BC→A, BC→D, D→E, D→F, D→C, F→D, F→E, G→F
Po odstránení redundantných funkčných závislostí získavame jedno z minimálnych pokrytí:
AB→C, AF→G, BC→A, BC→D, D→F, D→C, F→D, F→E, G→F

Iné minimálne pokrytia (možno sú aj ďalšie):
AB→C, AF→G, BC→A, BC→D, D→F, D→C, F→D, D→E, G→F

AB→C, AF→G, BC→A, BC→D, D→E, D→F, F→C, F→D, G→F

c) Dekomponujte r do tretej normálnej formy, bezstratovo a so zachovaním všetkých funkčných závislostí. **(1)**

3NF dekompozícia (z toho prvého minimálneho pokrytia):
{A, B, C}, {A, F, G}, {B, C, D}, {D, F}, {E, F}

Alebo (všetky atribúty vo funkčných závislostiach minimálneho pokrytia s rovnakou ľavou stranou môžeme dať do jednej relácie):
{A, B, C, D}, {A, F, G}, {D, E, F}

Alebo (všetky kľúčové atribúty môžeme dať do jednej relácie):
{A, B, C, D, F, G}, {B, E, F}

d) Dekomponujte r do Boyce-Coddovej normálnej formy, bezstratovo. Snažte sa vyhnúť zbytočnému rozbitiu funkčných závislostí. **(2)**

Začnime s 3NF dekompozíciou {A, B, C, D}, {A, F, G}, {D, E, F}.
V {A, F, G} porušuje BCNF G→F. V {A, B, C, D} porušuje BCNF D→C.
BCNF dekompozícia:
{A, B, D}, {A, G}, {C, D}, {D, E, F}, {F, G}

3. a) Definujte pojem zachovania funkčnej závislosti pri dekompozícii rel. schémy. **(1)**

Funkčná závislosť f je zachovaná v dekompozícii relačnej schémy (r, F) do
$(r_1, F_1), ..., (r_N, F_N,)$, ak f je dôsledkom zjednotenia funkčných závislostí $\cup_{i=1, ..., N} F_i$, t.j. ak $f \in (\cup_{i=1, ..., N} F_i)^+$, kde F_i označuje množinu funkčných závislostí, ktoré platia lokálne v r_i.

b) Vysvetlite, prečo je pri dekompozícii relačnej schémy rozumné zachovať funkčné závislosti. **(1)**

Funkčné závislosti je rozumné zachovať kvôli automatickej ochrane integrity (konzistencie) databázy.

Funkčná závislosť je integritné obmedzenie na databázu. Hoci norma SQL nepodporuje v DDL koncept funkčnej závislosti priamočiaro, funkčná závislosť sa dá v súčasných systémoch vyjadriť v podmienkach konštruktov CHECK, ASSERTION, TRIGGER a pod. Systém pri modifikácii databázy nedovolí integritu databázy porušiť (abortuje transakciu, ktorá sa o to pokúsi). Pri zachovaní funkčných závislostí je možné kontrolu robiť lokálne pre jednu reláciu. O čo lokálnejšia je kontrola, o to efektívnejšia môže byť.

c) Uvažujte relačnú schému z úlohy 2. Zapíšte funkčnú závislosť BC→AD ako formulu relačného kalkulu. **(2)**

$\forall B \forall C \forall A1 \forall D1 \forall E1 \forall F1 \forall G1 \forall A2 \forall D2 \forall E2 \forall F2 \forall G2$
$(r(A1, B, C, D1, E1, F1, G1) \wedge r(A2, B, C, D2, E2, F2, G2)) \Rightarrow (A1=A2 \wedge D1=D2)$

Ekvivalentne,
$\neg$ (
$\quad \exists B \exists C \exists A1 \exists D1 \exists E1 \exists F1 \exists G1 \exists A2 \exists D2 \exists E2 \exists F2 \exists G2$
$\quad r(A1, B, C, D1, E1, F1, G1) \wedge r(A2, B, C, D2, E2, F2, G2) \wedge \neg (A1=A2 \wedge D1=D2)$
)

V SQL sa dá táto funkčná závislosť vyjadriť napríklad takto:

```
create assertion r_fd_BC_AD
check not exists (
        select *
        from r r1, r r2
        where r1.B = r2.B and r1.C = r2.C and not (r1.A = r2.A and r1.D = r2.D)
)
```

4. a) Definujte triedu konflikt-sériovateľných rozvrhov. **(1)**

Rozvrh je konflikt-sériovateľný, ak jeho projekcia na commitované transakcie je konflikt-ekvivalentná niektorému sériovému rozvrhu tých commitovaných transakcií.

Dva rozvrhy sú konflikt-ekvivalentné, ak (obsahujú rovnaké operácie a zároveň) relatívne poradie všetkých konfliktných dvojíc operácií je v oboch rozvrhoch rovnaké.

Dve operácie (uvažujeme len kombinácie read a write) v rozvrhu sú konfliktné, ak sa týkajú rôznych transakcií, rovnakého objektu, a aspoň jedna z tých dvoch operácií je write.

b) Uvažujte triedu rozvrhov alt-serialisable definovanú takto: Rozvrh R patrí do triedy alt-serialisable, ak po jeho vykonaní je stav databázy rovnaký ako po vykonaní niektorého sériového rozvrhu S, pričom (S obsahuje rovnaké transakcie ako R a zároveň) transakcie v S sú vzostupne usporiadané podľa relatívneho poradia (t.j. časovej následnosti) ich prvých operácií v R. Porovnajte triedu konflikt-sériovateľných rozvrhov s triedou alt-serialisable (v zmysle množinovej inklúzie). Zdôvodnite. **(1)**

Uvažujme rozvrh R1: w1(X), w2(X), w1(X), c1, c2. Tento rozvrh nie je konflikt-sériovateľný, avšak po jeho vykonaní je stav databázy rovnaký ako po vykonaní sériového rozvrhu T2→T1. Teda R1∈alt-serialisable, avšak R1∉conflict-serialisable. Teda neplatí alt-serialisable ⊆ conflict-serialisable.

Uvažujme rozvrh R2: r1(X), w2(Y), w1(Y), c1, c2. Tento rozvrh je konflikt-sériovateľný, lebo je konflikt-ekvivalentný sériovému rozvrhu T2→T1. Avšak R2 nepatrí do triedy alt-serialisable, lebo R2 začína operáciou od T1 a po vykonaní sériového rozvrhu T1→T2 nie je stav databázy vo všeobecnosti rovnaký ako po vykonaní rozvrhu R2. Teda neplatí conflict-serialisable ⊆ alt-serialisable.

Triedy conflict-serialisable a alt-serialisable sú neporovnateľné.

c) Vysvetlite metódu časových pečiatok na implementáciu izolácie transakcií. **(2)**

Každá transakcia T dostane pri vykonaní svojej prvej operácie časovú pečiatku TS(T). Každý objekt X má 2 časové pečiatky, TS_R(X) a TS_W(X).

Keď T číta X, systém najskôr porovná TS(T) s TS_W(X). Ak TS(T)<TS_W(X), tak systém abortuje T. Inak, ak TS(T)>TS_R(X), tak TS_R(X) sa aktualizuje na hodnotu TS(T).

Keď T píše do X, systém najskôr porovná TS(T) s TS_W(X) aj s TS_R(X). Ak TS(T)<TS_W(X) alebo TS(T)<TS_R(X), tak systém abortuje T. Inak sa TS_W(X) sa aktualizuje na hodnotu TS(T).

Táto základná verzia metódy časových pečiatok generuje podtriedu konflikt-sériovateľných rozvrhov, ktorá sa dá rozšíriť opatrným zoslabením horeuvedených pravidiel (samozrejme, so zachovaním garancie izolácie transakcií).