9/1/2014 Úvod do databáz, **skúškový** test, max 25 bodov, 90 min

1. Daná je databáza (bez duplikátov a null hodnôt): capuje(Krcma, Alkohol, Cena), lubi(Pijan, Alkohol), navstivil(Idn, Pijan, Krcma), vypil(Idn, Alkohol, Mnozstvo).
Platí: Idn → Pijan, Krcma; Krcma, Alkohol → Cena; Idn, Alkohol → Mnozstvo;
Mnozstvo > 0; Cena > 0.
a) Nájdite trojice [P1, P2, K] také, že pijani P1 aj P2 krčmu K niekedy navštívili, ale (nikdy) v K nevypili (žiaden) alkohol, ktorý obaja ľúbia. Sformulujte tento dotaz v Datalogu **(2),** relačnom kalkule **(2)** SQL **(2)** a relačnej algebre **(2)**.

Presnejšia formulácia dotazu v hovorovom jazyku: Nájdite trojice [P1, P2, K] také, že pijani P1 aj P2 krčmu K niekedy navštívili; a pre žiaden z alkoholov, ktorý P1 aj P2 ľúbia, sa nestalo, že P1 aj P2 ten alkohol vypili v krčme K.

Relačný kalkul:

$$\{[P1, P2, K]:$$
$$\exists I1\ \exists I2$$
$$navstivil(I1, P1, K) \wedge navstivil(I2, P2, K) \wedge$$
$$\neg ($$
$$\exists A\ \exists I1\ \exists I2\ \exists M1\ \exists M2$$
$$lubi(P1, A) \wedge navstivil(I1, P1, K) \wedge vypil(I1, A, M1) \wedge$$
$$lubi(P2, A) \wedge navstivil(I2, P2, K) \wedge vypil(I2, A, M2)$$
$$)$$
$$\}$$

Datalog:

```
?- answer(P1, P2, K).

answer(P1, P2, K) ←
        navstivil(_, P1, K),
        navstivil(_, P2, K),
        not vypili_spolocny(P1, P2, K).

vypili_spolocny(P1, P2, K) ←
        lubi(P1, A),
        navstivil(I1, P1, K),
        vypil(I1, A, _),
        lubi(P2, A),
        navstivil(I2, P2, K),
        vypil(I2, A, _).
```

SQL:

```
create temporary table vypili spolocny as
select n1.Pijan as P1, n2.Pijan as P2, n.Krcma as K
from lubi l1, navstivil n1, vypil v1, lubi l2, navstivil n2, vypil v2
where l1.Pijan = n1.Pijan and n1.Idn = v1.Idn and l1.Alkohol = v1.Alkohol and
        l2.Pijan = n2.Pijan and n2.Idn = v2.Idn and l2.Alkohol = v2.Alkohol)

/* answer */
select n1.Pijan as P1, n2.Pijan as P2, n.Krcma as K
from navstivil n1, navstivil n2
where n1.Krcma = n2.Krcma and not exists (
        select *        /* P1 niekedy vypili spolocny oblubeny alkohol v K */
        from vypili spolocny vs
        where n1.Pijan = vs.P1 and n1.Krcma = vs.K and n2.Pijan = vs.P2)
```

Relačná algebra:

$$\text{vypili\_spolocny} := \Pi_{P1 \leftarrow lnv1.Pijan,\ P2 \leftarrow lnv2.Pijan,\ lnv1.Krcma} \left(P_{lnv1}(\text{lubi} \bowtie \text{navstivil} \bowtie \text{vypil}) \bowtie_{lnv1.Krcma = lnv2.Krcma \wedge lnv1.Alkohol = lnv2.Alkohol} P_{lnv2}(\text{lubi} \bowtie \text{navstivil} \bowtie \text{vypil}) \right)$$

/* answer */

$$\Pi_{P1 \leftarrow n1.Pijan,\ P2 \leftarrow n2.Pijan,\ n1.Krcma} (P_{n1}(\text{navstivil}) \bowtie_{n1.Krcma = n2.Krcma} P_{n2}(\text{navstivil})) - \text{vypili\_spolocny}$$

Nejednoznačnosť hovorového jazyka (nielen slovenčiny) umožňuje tento dotaz interpretovať aj inak: Nájdite trojice [P1, P2, K] také, že pijani P1 aj P2 krčmu K niekedy navštívili; a pre žiaden z alkoholov, ktorý P1 aj P2 ľúbia, sa nestalo, že niektorý z P1 alebo P2 ten alkohol vypil v krčme K.

Relačný kalkul:

$$\{[P1, P2, K]: \exists I1\ \exists I2\ navstivil(I1, P1, K) \wedge navstivil(I2, P2, K) \wedge \neg ((\exists A\ \exists I\ \exists M\ lubi(P1, A) \wedge lubi(P2, A) \wedge navstivil(I, P1, K) \wedge vypil(I, A, M)) \vee (\exists A\ \exists I\ \exists M\ lubi(P1, A) \wedge lubi(P2, A) \wedge navstivil(I, P2, K) \wedge vypil(I, A, M))) \}$$

Datalog:

```
?- answer(P1, P2, K).

answer(P1, P2, K) ←
        navstivil(_, P1, K),
        navstivil(_, P2, K),
        not niekto_vypil_spolocny(P1, P2, K).

niekto_vypil_spolocny(P1, P2, K) ←
        lubi(P1, A),
        lubi(P2, A),
        navstivil(I, P1, K),
        vypil(I, A, _).

niekto_vypil_spolocny(P1, P2, K) ←
        lubi(P1, A),
        lubi(P2, A),
        navstivil(I, P2, K),
        vypil(I, A, _).
```

SQL:

```
create temporary table niekto_vypil_spolocny as
select l1.Pijan as P1, l2.Pijan as P2, n.Krcma
from lubi l1, lubi l2, navstivil n, vypil v
where l1.Alkohol = l2.Alkohol and l1.Alkohol = v.Alkohol and
        n.Idn = v. Idn and (n.Pijan = l1.Pijan or n.Pijan = l2.Pijan)

/* answer */
select n1.Pijan as P1, n2.Pijan as P2, n.Krcma as K
from navstivil n1, navstivil n2
where n1.Krcma = n2.Krcma and not exists (
        select *
        from niekto_vypil spolocny nvs
        where n1.Pijan = nvs.P1 and n1.Krcma = nvs.K and n2.Pijan = nvs.P2)
```

Relačná algebra:

$$niekto\_vypil\_spolocny := \Pi_{P1 \leftarrow lnv.Pijan,\ P2 \leftarrow l.Pijan,\ lnv.Krcma} \left(P_{lnv} (lubi \bowtie navstivil \bowtie vypil) \bowtie_{lnv.Alkohol = l.Alkohol} P_{l} (lubi) \right) \cup \Pi_{P1 \leftarrow l.Pijan,\ P2 \leftarrow lnv.Pijan,\ lnv.Krcma} \left(P_{lnv} (lubi \bowtie navstivil \bowtie vypil) \bowtie_{lnv.Alkohol = l.Alkohol} P_{l} (lubi) \right)$$

/ answer */*

$$\Pi_{P1 \leftarrow n1.Pijan,\ P2 \leftarrow n2.Pijan,\ n1.Krcma} (P_{n1} (navstivil) \bowtie_{n1.Krcma = n2.Krcma} P_{n2} (navstivil)) - niekto\_vypil\_spolocny$$

b) Uvažujte databázu rozšírenú o reláciu vyraba(Alkohol, Firma), kde Alkohol → Firma. Sformulujte v Datalogu **(2)** a SQL **(2)** dotaz na krčmy, v ktorých obrat z alkoholov od firmy oldherold je menší než 10% celkového obratu firmy oldherold (cez všetky krčmy).

Do výsledku patria aj krčmy, kde sa alkohol od oldherold nevypil.

Datalog:

```
?- answer(K).

answer(K) ←
        oh_total(T),
        oh_krcma(K, S),
        S < 0.1 * T. /* F is 0.1 * T, S < F */

oh_total(T) ←
        subtotal(ucet_oh(_, _, U), [], [T = sum(U)]).

oh_krcma(K, S) ←
        subtotal(ucet_oh(_, K, U), [K], [S = sum(U)]).

obrat_oh_krcma(K, 0) ←
        capuje(K, _, _),
        not ucet_oh(_, K, _).

ucet_oh(I, K, U) ←
        navstivil(I, _, K),
        vypil(I, A, M),
        vyraba(A, oldherold),
        capuje(K, A, C),
        U = M * C. /* U is M * C */
```

SQL:

```
create temporary table ucet_oh as
select n.Idn, n.Krcma, v.Mnozstvo * c.Cena as U
from navstivil n, vypil v, vyraba f, capuje c
where n.Idn = v.Idn and n.Krcma = c.Krcma and v.Alkohol = f.Alkohol and v.Alkohol = c.Alkohol and
        f.Firma = ‘oldherold‘

create temporary table oh_total as
select sum(u.U) as T
from ucet_oh u

create temporary table oh_krcma as
select u.Krcma, sum(u.U) as S
from ucet_oh u
group by u.Krcma
union
select c.Krcma, 0 as S
from capuje c
where not exists (
        select *
        from ucet_oh u
        where c.Krcma = u.Krcma)

/* answer */
select k.Krcma
from oh_total t, oh_krcma k
where k.S < 0.1 * T
```

2. Daná je relácia r(A, B, C, D, E, F, G, H) s funkčnými závislosťami
BE→GH, BEG→FA, D→C, F→B, BF→A.
a) Nájdite všetky kľúče relácie r. **(1)**

Atribúty D a E sú v každom kľúči, lebo nie sú na pravej strane žiadnej funkčnej závislosti.
Atribút C nie je v žiadnom kľúči, lebo D→C.

ABDEFGH
-A: BDEFGH
-B: DEFGH
-F: DEGH
+F: **DEF**
+B: **BDE**
+A: ADEGH

Kľúče v r sú BDE a DEF, iné kľúče v r nie sú.

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

Po minimalizácii ľavých strán kánonických funkčných závislostí:
BE→G, BE→H, BE→F, BE→A, D→C, F→B, F→A

Po odstránení redundantných funkčných závislostí získame minimálne pokrytie:
BE→G, BE→H, BE→F, D→C, F→B, F→A

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

r1(A, F), r2(B, E, G), r3(B, E, H), r4(B, E, F), r5(C, D), r6(D, E, F)

Alternatívne riešenie. Všetky atribúty vo funkčných závislostiach minimálneho pokrytia s rovnakou ľavou stranou môžeme dať do jednej relácie (to sa dá urobiť vždy, nielen v tomto konkrétnom prípade). Do relácie (B, E, F, G, H) môžeme pridať tiež atribút D bez porušenia 3NF.
r1(A, B, F), r2(B, D, E, F, G, H), r3(C, D)

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

Začneme s 3NF z alternatívneho riešenia.
r3 je binárna relácia, teda je v BCNF. V r1 platí F→B, avšak F je nadkľúč v r1. Teda r1 je v BCNF. V r2 platia rôzne netriviálne závislosti, ktoré porušujú BCNF kvôli kľúčovým atribútom D a E (ktoré sa nedajú určiť z ostatných atribútov). Začnime teda dekompozíciou r2 do (D, E, F) (ktorá je v BCNF) a (B, E, F, G, H) (ktorá je v 3NF). V (B, E, F, G, H) platí BE→G, BE→H, BE→F a F→B, ktoré porušujú BCNF. Postupnou dekompozíciou podľa týchto funkčných závislostí dostaneme (B, E, G), (B, E, H), (B, F), (B, E), ktoré už ďalej dekomponovať netreba, lebo sú v BCNF. Relácia (B, E) je podmnožinou (B, E, H), takže ju možno vynechať. Podobne, relácia (B, F) je podmnožinou (A, B, F), takže ju možno vynechať.

BCNF dekompozícia:
r1(A, B, F), r2(D, E, F), r3(B, E, G), r4(B, E, H), r5(C, D)

3. Uvažujte nasledujúce dva SQL dotazy nad reláciou r(A, B).
Q1: select distinct A from r group by A, B having A=B
Q2: select A from r where A=B group by A
Zapíšte Q1 a Q2 v rel. algebre (bez optimalizácií) **(2)** a rozhodnite, či sú ekvivalentné **(1)**.

V relačnej algebre:
Q1: $\Delta(\pi_A(\sigma_{A=B}(\Gamma_{A,B}(r))))$
Q2: $\pi_A(\Gamma_A(\sigma_{A=B}(r)))$

Algebraické výrazy Q1 a Q2 predstavujú rôzne výpočty dotazov Q1 a Q2. To však neznamená, že ich výsledky sú rôzne.

Upravujme ekvivalentne (vzhľadom na výsledok výpočtu) algebraický výraz pre Q1. V operátore Γ sa nerobí žiadna agregácia, takže výsledkom $\Gamma_{A,B}(r)$ je vynechanie duplikovaných záznamov r (ktoré sa zhodujú na A aj na B). Teda
$Q1 \equiv \Delta(\pi_A(\sigma_{A=B}(\Delta(r))))$
Duplikáty pri výpočte Q1 výsledok neovplyvňujú (v Q1 sa nerobí žiadna agregácia), stačí ich odstrániť až nakoniec:
$Q1 \equiv \Delta(\pi_A(\sigma_{A=B}(r)))$

Upravujme ekvivalentne (vzhľadom na výsledok výpočtu) algebraický výraz pre Q2. V operátore Γ sa nerobí žiadna agregácia, takže výsledkom $\Gamma_A(r)$ je projekcia r na atribút A s následným vynechaním duplikovaných záznamov. Teda
$Q2 \equiv \pi_A(\Delta(\pi_A(\sigma_{A=B}(r))))$
Finálna projekcia nemá žiaden efekt a dá sa vynechať:
$Q2 \equiv \Delta(\pi_A(\sigma_{A=B}(r)))$

Teda **Q1 ≡ Q2**.

4. Niektoré transakčné systémy používajú okrem read- a write-locks aj tzv. upgrade-lock. Upgrade-lock sa používa a správa ako read-lock (transakcii, ktorá ho vlastní, dáva iba právo na čítanie zamknutých dát). Avšak transakcia, ktorý vlastní upgrade-lock na nejaký objekt, smie požiadať o write-lock na ten objekt (presnejšie, o zmenu svojho upgrade-lock na write-lock). Okrem bežných pravidiel pre prideľovanie zámkov platia nasledujúce pravidlá. Keď transakcia T1 vlastní write-lock na objekt X, nesmie systém prideliť inej transakcii T2 upgrade-lock na X, a naopak. Keď transakcia T1 vlastní read-lock na objekt X, tak systém smie prideliť inej transakcii T2 upgrade-lock na X (ale nie naopak—t.j. keď T1 vlastní upgrade-lock na X, nesmie systém prideliť inej transakcii T2 read-lock na X).
a) Uveďte rozvrh dvoch commitovaných transakcií, ktorý je možné generovať striktným 2PL s upgrade-locks, ale nie je možné ho generovať striktným 2PL bez upgrade-locks (ten rozvrh nemá obsahovať operácie, ktoré sa týkajú zamykania). Vysvetlite. **(2)**

Takým rozvrhom je napríklad **r1(A), r2(A), c1, w2(A), c2**.
Rozvrh s upgrade-locks: rlock1(A), r1(A), upglock2(A), r2(A), unlock1(A), c1, wlock2(A), w2(A), unlock2(A), c2.
Tento rozvrh sa nedá generovať s 2PL bez upgrade locks, lebo už pred čítaním r2(A) potrebuje T2 získať write-lock. Lenže ten vtedy (pred commitom T1) získať nemôže, lebo je v konflikte s read-lock, ktorý drží T1.

b) Uveďte rozvrh dvoch commitovaných transakcií, ktorý demonštruje nutnosť asymetrie vyjadrenej dodatkom "ale nie naopak…". Vysvetlite. **(2)**

Cieľom 2PL je garancia konflikt-sériovateľnosti generovaných rozvrhov. Riziko zmeny pravidiel 2PL spočíva v porušení tejto garancie. Treba overiť, či rozšírenie 2PL o upgrade-lock bez toho dodatku garanciu konflikt-sériovateľnosti poruší.

V 2PL bez upgrade-lock je každý prefix generovaného rozvrhu konflikt-sériovateľný. Kompletný generovaný rozvrh je konflikt-ekvivalentný sériovému rozvrhu, v ktorom transakcie commitujú v poradí prvého uvoľnenia niektorého zámku.

Keďže upgrade-lock sa správa ako read-lock, k porušeniu konflikt-sériovateľnosti generovaného rozvrhu môže dôjsť až po zmene upgrade-lock na write-lock. (Zmenu upgrade-lock na write-lock system povolí len keď žiadna iná transakcia vtedy nedrží zámok na ten objekt.)
Predpokladajme, že vo výslednom rozvrhu nejaká transakcia T1 prvýkrát úspešne mení svoj upgrade-lock na write-lock na nejakom objekte X. Do toho momentu sa každý upgrade-lock sa správal ako read-lock, teda rozvrh bol generovaný podľa štandardného 2PL. Pre transakciu T2, ktorá je v generovanom rozvrhu v konflikte s T1 (read-write, write-read, write-write), sú v tomto momente iba dve možnosti:
1. Transakcia T2 niekedy predtým uvoľnila nejaký svoj zámok. To znamená, že T2 už o žiaden ďalší zámok žiadať nebude. Na žiadnom z objektov, ktoré má T2 v tomto momente zamknuté, sa T2 nikdy nedostane do konfliktu typu T1→T2. Teda rozvrh bude konflikt-sériovateľný.
2. Transakcia T2 dosiaľ neuvoľnila žiaden zámok. Z toho vyplýva, že T2 nikdy nevlastnila zámok na X. Teda je to tak, ako keby T1 v minulosti namiesto o upgrade-lock požiadala rovno o write-lock. Tento upgrade-lock teda nespôsobí problém s konflikt-sériovateľnosťou a celý argument vieme zopakovať pre nasledujúci upgrade-lock.

Z toho vyplýva, že 2PL rozšírený o upgrade-lock horeuvedeným spôsobom generuje konflikt-sériovateľné rozvrhy (dokonca každý prefix generovaného rozvrhu je konflikt-sériovateľný)—aj bez toho dodatku, aj s ním. Ten dodatok nie je nutný (je priam zbytočne reštriktívny).

c) Ako zareaguje scheduler systému, keď dostane od transakcie T2 žiadosť o read-lock na X, pričom iná transakcia T1 v tom momente drží upgrade-lock na X? **(1)**

Ak scheduler nepoužíva žiadnu metódu prevencie proti deadlocku, tak odloží vykonanie operácie rl2(X) na neskôr, t.j. žiadosť transakcie T2 o read-lock na X nechá čakať.
Ak scheduler používa nejakú metódu prevencie proti deadlocku, tak najskôr spustí tú metódu a až potom sa venuje tej žiadosti o read-lock.