Príklad 1.b.

Zapíšte v slovenskom jazyku **(1)**, v Datalogu **(2)** a v SQL **(2)**:

{P: ∃A'∃I'∃K' (lubi(P, A') ∨ navstivil(I', P, K')) ∧

∀K∀A∀I (((¬ capuje(K, A) ∨ lubi(P, A)) ∧ ¬ navstivil(I, P, pizzahut)))}

Upravená ekvivaletná formula:

{P: ∀(A,K,I)((capuje(K, A) ⇒ lubi(P, A)) ∧ ¬ navstivil(I, P, "pizzahut")}

Po slovensky:

Pijani, ktorí ľúbia všetky alkoholy, ktoré niekde čapujú a nikdy nenavštívili krčmu "pizzahut".

Datalóg:

pijan(P) ← lubi(P, _). /* Pijani, ktorí neľúbia žiaden alkohol sa nepodielajú na výsledku. Okrem degenerovaného prípadu, keď krčmy neslúžia svojmu účelu. */

nevyhovuje(P) ← pijan(P), capuju(_, A), ¬ lubi(P, A).

vyhovuje(P) ← pijan(P), ¬ nevyhovuje(P), ¬ navstivil(_, P, "pizzahut").

Môžeme pijana rovno nahradiť pravou stranou. Vzniklý program je bezpečný. Pozri nasledujúci príklad.

nevyhovuje(P) ← lubi(P, _), capuju(_, A), ¬ lubi(P, A).

vyhovuje(P) ← lubi(P, _), ¬ nevyhovuje(P), ¬ navstivil(_, P, "pizzahut").

SQL:

```
select L.Pijan from lubi L
where ( not exists ( select  Lu.Pijan  from  lubi Lu,  capuju Ca
                     where ( Lu.Pijan =  L.Pijan and
                          not exists (select  * from lubi Lubi
                                      where ( Lubi.Pijan =  Lu.Pijan and
                                              Lubi.Alkohol = Ca Alkohol ))))
        and not exists  ( select  Na.Pijan  from navstivil Na
                          where ( Na.Pijan =  L.Pijan and
                                  Na.Krcma = "pizzahut" )));
```

To je správne riešenie v SQL. Všetky create view alebo with alebo dotazy za from sú prznením SQL. Vlastne bez nenavštíveného pizzahutu je čiste relačné delenie. Pôvodny Codov príklad.
To dvojité not exists (not in) sa učí aj na komerčných kurzoch SQL a kladie sa dôraz na to, aby to frekventanti vedeli.

Príklad 2.

Definujte bezpečnosť programov v Datalogu. **(2)**
Vysvetlite, prečo sa od Datalogových programov vyžaduje, aby boli bezpečné. **(1)**

Najprv definícia:

Datalógový program je bezpečný, keď všetky jeho pravidlá sú bezpečné.
Pravidlo je bezpečné, keď všetky premenné sú bezpečné a každý negovaný predikát obsahuje aspoň jednu premennú, ktorá sa v tele pravidla vyskytuje aj v nejakom nenegovanom predikáte.

Premenná je bezpečná keď

i.) sa vyskytuje v tele v pozitivnom nezabudovanom predikate.
ii.) nadobuda hodnoty z konečnej množiny konštánt ($x=a \vee x=b \vee ...$)
iii.) rovná sa inej bezpečnej premennej ($x=y$ a y bezpečná).

Túto definíciu môžeme trochu oslabiť tým, že budeme požadovať bezpečnosť len pre významné premenné (sú to premenné, ktoré sa vyskytujú v hlave pravidla; alebo sa vyskytujú v tele pravidla viackrát – nie podčiarovníky). Nevýznamné premenné nemusia byť bezpečné.

Príklad: $a(X, Y) \leftarrow p(X, Y), \neg q(Y, Z).$

Je naprosto korektné pravidlo. Jeho význam je $\{<X, Y>: p(X,Y) \wedge \neg (\exists Z) q(Y, Z)\}$.

V relačnej algebre je to antijoin, množina tých n-tíc P(X, Y), ktoré sa prirodene nespájajú s Q(Y, Z). $A(X, Y) = P(X, Y) - \Pi_{X,Y} P(X, Y) \bowtie Q(Y, Z).$

Ekvivalentný aj formálne bezpečný program je :

$n(X, Y) \leftarrow p(X, Y), q(Y, Z).$

$a(X, Y) \leftarrow p(X, Y), \neg n(X, Y).$

Bezpečnosť formúl je to, čo vydeluje teóriu databáz z matematickej teórie modelov a odlišuje ju od algebry.

Skúmanie modelov množín formúl (teórii) je rozvinutá disciplína algebry. Matematikov zaujíma prevažne aký dopad majú formuly na štruktúru oborov definície (je to torzná grupa, pole, cylindrická algebra a pod.). V databázach nás zaujímajú iba vlastnosti, vyplývajúce iba z obsahu relácii (tabuliek), ktoré nezávisia na struktúre domén. Zaujímajú nás doménovo nezávislé formuly. T.j. také formuly, ktorých pravdivosť a nepravdivosť, množina ohotnotení pre ktoré sú pravdivé závisí len na naplnení relácii a nie na obsahu domén.

Či formula je doménovo nezávislá je nerozhodnuteľné (M. Vardi). Bezpečné formuly sú doménovo nezávislé (vlastná podmnožina). Ako sme videli bezpečnosť sa ľahko testuje. Stačí overiť vlastnosti i, ii, iii, pre všetky podformuly (datalógový program).

Bezpečnosť zaručuje, že keď pracujeme s konečnou databázou, výsledky a medzivýsledky budú vždy konečné a dokážeme ich napísať. Nezávisle na tom, aké sú domény.

Príklad 1.a.

Daná je databáza
capuje(Krcma, Alkohol)

lubi(Pijan, Alkohol)

navstivil(Idn, Pijan, Krcma)

vypil(Idn, Alkohol, Mnozstvo)

a) Sformulujte nasledujúci dotaz v relačnom kalkule **(2)**, relačnej algebre **(2)**, Datalogu **(2)** a SQL **(2)**: Nájdite dvojice [Alkohol, Mnozstvo], ktoré o každom alkohole hovoria, aké množstvo ho bolo vypité s nechuťou, t.j. aké celkové množstvo toho alkoholu vypili pijani, ktorí ten alkohol neľúbia. Nájsť treba všetky alkoholy, ktoré sa čapujú v niektorej krčme.

Kalkul:

$$\Psi(I, Q = sum(M))(((\exists K)(capuju(K, A) \wedge I=0 \wedge M=0)) \vee ((\exists K)(navstivil(I, P, K) \wedge vypil(I, A, M) \wedge \neg lubi(P, A))))$$

Algebra:

$$\Gamma_{A,Q=sum(M)}(\Pi_{A,I,M} capuju \times \{< I=0, M=0>\} \cup \Pi_{A,I,M}(navstivil \bowtie vypil) - \Pi_{A,I,M}(navstivil \bowtie vypil \bowtie lubi))$$

Datalóg:

```
vypite_s_nechutou(A, 0, 0) ← capuju( _, A). /* inicializuje všetky alkoholy na 0 */
vypite_s_nechutou(A, I, M) ← navstivil(I, P, _ ), vypil(I, A, M), ¬ lubi(P, A).
odpoved(A, Q) ← subtotal( vypite_s_nechutou(A, I, M), A, Q = sum(M)).
```

SQL:

```
select V.A, sum(V.M) as Q  from vypil V, navstivil N
where (V.I = N.I and not exists
            (select * from lubi L where (L.P = N.P and L.A = V.A)
group by V.A
union
select C.A , 0 as Q from capuju C
where not exits ( select N.I from vypil V, navstivil N
                  where (C.A = V.A and V.I = N.I and not exists
                        (select * from lubi L where (L.P = N.P and L.A = V.A));
```

V SQL sa väčšina dotazov dá realizovať pomocou vložených dotazov za where. Netreba používať príkazy create alebo with, či dotaz za from. V niektorých prípadoch nám však takéto dotazy ušetria veľa písania.

Podstatným rozšrením je príkaz with recursive, ktorý rozširuje SQL o operátor

najmenšieho pevného bodu (iteráciu).