Databázové systémy

Stano Krajči

Ústav informatiky
Prírodovedecká fakulta
UPJŠ Košice

2005

Predslov

Vážení čitatelia.

Zámerom tohto textu je poskytnúť študentom informatiky na Prírodovedeckej fakulte UPJŠ v Košiciach materiál na základné štúdium databázových systémov. Je to kompromis medzi strohým matematickým formalizmom, používateľskou príručkou a voľným rozprávaním. Drží sa zásady **Exempla trahunt** – celým textom sa tiahne jeden príklad z vysokoškolákom dobre známej oblasti. Na ňom sú postupne vysvetľované základné i podružné databázové pojmy a princípy. Študent sa tak krok za krokom viac či menej podrobne zoznámi so všetkými podstatnými aspektmi práce databázistu – od základného porozumenia jazyka SQL cez databázové modelovanie (hlavne podkapitola 2.4), až po teoretické základy databáz (najmä 4. kapitola).

Treba si však uvedomiť, že materiálom v tomto texte nie je problematika databázových systémov ani zďaleka vyčerpaná. Je to jednoducho písomná forma dvojsemestrálneho vysokoškolského kurzu. Niektorých vecí sa naozaj dotkneme len zľahka, no prípadnému hlbšiemu záujemcovi nič nebráni, aby sa chopil ďalších učebníc, odborných článkov či, na druhej strane, manuálov príslušného databázového jazyka, a tak svoje vedomosti ďalej zdokonaľoval.

Aj keď je text orientovaný takmer výlučne na databázový produkt DB2, princípy v ňom uvádzané majú širšiu platnosť. Som presvedčený, že pre čitateľa, ktorý sa (aj prostredníctvom tohto textu) dostatočne oboznámi s týmto kvalitným databázovým systémom, je prechod na iný produkt záležitosťou niekoľkých dní.

Dovoľte mi na tomto mieste pár slov vďaky. Najväčšia patrí mojim priateľom a exkolegom z firmy VSL Software (a špeciálne Peťovi Bugatovi), ktorí ma donútili rozšíriť si obzor a spriateliť sa s databázami. Nemôžem tiež zabudnúť na podnetné električkové rozhovory s Peťom Mihókom a Gabikou Andrejkovou. Ďakujem aj svojim kolegom Romanovi Sotákovi, ktorý mi v ťažkých začiatkoch nezištne poskytol svoje písomné prípravy, a Hamovi Hámorskému a Maťovi Nemčekovi za námety na zaujímavé úlohy. Oceňujem aj prvých oficiálnych čitateľov – recenzentov – Peťa Mihóka a Vlada Žoldáka, ktorí mi tak obetovali riadne kusisko svojho vzácneho času.

V neposlednom rade sa musím poďakovať aj študentom, ktorí svojimi otázkami na prednáškach i mimo nich posunuli tento učebný text ďalej. Za úspešnú možno považovať aj hru **Hľadaj chybu**, keď ma študenti (nie celkom nezištne) upozornili na desiatky chýb typografického, ortografického i odborného charakteru. Pevne (hoci márne) dúfam, že odteraz už žiadnu nenájdu ;-).

A teraz Vás už pozývam na stretnutie s databázami...

Stano Krajči

Obsah

0 Úvod . . . 9
0.1 História databázových technológií . . . 10
0.2 História SQL . . . 11
0.3 Princípy databázových systémov a SQL . . . 12

1 Práca s jednou tabuľkou . . . 13
1.1 Každý začiatok je – ľahký . . . 14
1.1.1 Prieskum terénu . . . 14
1.1.2 Vytvorenie tabuľky (CREATE TABLE, dátové typy, INSERT) . . . 14
1.2 Výpisy všetkých záznamov (SELECT) . . . 18
1.2.1 Základný výpis (ORDER BY, VALUES) . . . 18
1.2.2 Iba niektoré stĺpce . . . 21
1.2.3 Stĺpcové funkcie (||, SUBSTR, CAST, COALESCE, CASE, DAYS, ...) . . . 26
1.2.Ú Úlohy . . . 38
1.3 Výpisy záznamov spĺňajúcich nejakú podmienku (WHERE) . . . 39
1.3.1 Základné porovnania . . . 39
1.3.2 Zložitejšie porovnania a žolíky (BETWEEN, IN, LIKE) . . . 42
1.3.3 Logické spojky (NOT, AND, OR) . . . 44
1.4 Agregácie . . . 46
1.4.1 Jednoduché agregácie (COUNT, DISTINCT, MAX, MIN, AVG, SUM) . . . 46
1.4.2 Agregácie po skupinách (GROUP BY, CUBE, HAVING) . . . 49
1.5 Vnorené dopyty . . . 53
1.5.1 Vnorené dopyty bez parametra . . . 53
1.5.2 Vnorené dopyty s parametrom . . . 54
1.6 Manipulácia s dátami tabuľky . . . 59
1.6.1 Vkladanie záznamov (INSERT) . . . 59
1.6.2 Úprava záznamov (UPDATE) . . . 61
1.6.3 Mazanie záznamov (DELETE) . . . 66
1.6.4 Preddefinovaná hodnota (DEFAULT) . . . 67
1.6.Ú Úlohy . . . 69
1.7 Primárny kľúč a ďalšie integritné obmedzenia . . . 70
1.7.1 Primárny kľúč (PRIMARY KEY, NOT NULL) . . . 70
1.7.2 Sekundárny kľúč (UNIQUE) . . . 76
1.7.3 Kontrola (CHECK) . . . 78

2 Práca s viacerými tabuľkami . . . 81
2.1 Spojenie dvoch tabuliek . . . 82
2.1.1 Jedna tabuľka nestačí . . . 82
2.1.2 Spojenie (JOIN) . . . 85
2.1.3 Vonkajšie spojenia (LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN) . . . 89
2.2 Cudzí kľúč (REFERENCES, FOREIGN KEY) . . . 94
2.2.1 Načo je cudzí kľúč? . . . 94
2.2.2 Správanie cudzieho kľúča pri vkladaní dát . . . 96
2.2.3 Správanie cudzieho kľúča pri modifikácii dát . . . 96
2.2.4 Tri typy správania cudzieho kľúča pri mazaní dát (ON DELETE CASCADE, ON DELETE SET NULL) . . . 97
2.2.5 Správanie cudzieho kľúča pri mazaní odkazovanej tabuľky . . . 101
2.3 Modifikácia štruktúry tabuľky (ALTER TABLE) . . . 102
2.3.1 Pridávanie a modifikácia stĺpcov (ADD COLUMN, ALTER COLUMN) . . . 102
2.3.2 Pridávanie a mazanie integritných obmedzení (CONSTRAINT) . . . 105

2.4 Návrh väčšej databázovej štruktúry . . . 107
2.4.1 Konceptuálne modelovanie . . . 107
2.4.2 Transformácia modelu do databázy . . . 115
2.4.3 Revízia . . . 121
2.5 Množinové operácie . . . 125
2.5.1 Zjednotenie (UNION, UNION ALL) . . . 125
2.5.2 Prienik a rozdiel (INTERSECT, EXCEPT) . . . 130
2.5.3 Ich kombinácie . . . 131
2.5.4 Kvantifikátory (SOME, ANY, ALL, EXISTS) . . . 132
2.5.Ú Úlohy . . . 134
2.6 Ako postupovať pri konštrukcii dopytu? . . . 135
2.6.1 Model ako grafická pomôcka . . . 135
2.6.2 Pomocné tabuľky (WITH) . . . 139
2.6.3 Pomocné tabuľky a ich pomocné tabuľky . . . 142
2.6.4 Viac exemplárov tej istej tabuľky . . . 148
2.6.5 Ešte jeden odvodený údaj . . . 151
2.6.Ú Úlohy . . . 153

3 Ďalšie črty SQL . . . 155
3.1 Tranzitívny uzáver a rekurzia . . . 156
3.1.1 Tranzitívny uzáver . . . 156
3.1.2 Vyjadrenie tranzitívneho uzáveru pomocou rekurzie . . . 160
3.1.Ú Úlohy . . . 164
3.2 Pohľady (CREATE VIEW, DROP VIEW) . . . 165
3.2.1 Pojem pohľadu . . . 165
3.2.2 Hierarchické pohľady . . . 167
3.2.3 Rekurzívne pohľady . . . 168
3.2.4 Ovplyvňovanie tabuľky skrz pohľad . . . 168
3.2.5 Mazanie pohľadu . . . 173
3.2.6 Načo ešte sú pohľady? . . . 173
3.3 Triggery (CREATE TRIGGER, DROP TRIGGER) . . . 174
3.3.1 Ako funguje trigger? . . . 174
3.3.2 Triggery signalizujúce chybu (SIGNAL SQLSTATE) . . . 177
3.3.3 Triggery so superakciou (ATOMIC) . . . 179
3.3.Ú Úlohy . . . 179
3.4 Indexy (CREATE INDEX, DROP INDEX, CLUSTER) . . . 181
3.5 Systémové tabuľky . . . 183
3.6 Základy administrácie databázy . . . 186
3.6.1 Vytvorenie a zrušenie databázy (CREATE DATABASE, DROP DATABASE) . . . 186
3.6.2 Prístupové práva (GRANT, REVOKE) . . . 186
3.6.3 Spojenie s databázou (CONNECT, DISCONNECT) . . . 186
3.6.4 Zálohovanie (BACKUP DATABASE, RESTORE DATABASE) . . . 186
3.7 Spolupráca SQL a iných jazykov . . . 188
3.7.1 Embedded SQL . . . 188
3.7.2 Používateľom definované funkcie (CREATE FUNCTION, DROP FUNCTION) . . . 190

4 Teoretické základy databáz . . . 191
4.1 Formalizácia tabuľky . . . 192
4.1.1 Formalizácia stĺpca . . . 192
4.1.2 Abstraktná tabuľka . . . 192
4.1.3 Typovaná relácia . . . 193
4.1.4 Reálna tabuľka . . . 194
4.1.5 Zhrnutie . . . 195

4.2 Databázová logika . . . 196
4.2.1 Typované funkcie . . . 196
4.2.2 Termy . . . 197
4.2.3 Podmienky . . . 198
4.3 Relačná algebra . . . 201
4.3.1 Transformácia stĺpcov, projekcia a premenovanie stĺpcov . . . 201
4.3.2 Selekcia riadkov . . . 203
4.3.3 Spojenia dvoch tabuliek, karteziánsky súčin . . . 203
4.3.4 Zjednotenie, prienik, rozdiel . . . 207
4.3.5 Skladanie operácií . . . 209
4.3.Ú Úlohy . . . 216
4.4 Funkčné závislosti a normálne formy . . . 217
4.4.1 Funkčné závislosti . . . 217
4.4.2 Schéma abstraktných tabuliek . . . 218
4.4.3 Pokrytie . . . 221
4.4.4 Nadkľúč a kľúč . . . 222
4.4.5 1NF, 2NF, 3NF a BCNF . . . 223
4.4.6 Multizávislosti a 4NF . . . 227
4.4.7 Prirodzené spojenie, projekcia a bezstratová dekompozícia . . . 229
4.4.8 Dekompozícia schém . . . 234
4.4.Ú Úlohy . . . 238

A Apendix . . . 239
A.1 Riešenia niektorých úloh . . . 240
1.2-1 . . . 240
1.6-1 . . . 240
2.5-1 . . . 241
2.5-2 . . . 243
2.6-1 . . . 246
2.6-2 . . . 247
2.6-3 . . . 249
2.6-4 . . . 250
3.1-1 . . . 251
3.1-2 . . . 252
3.1-3 . . . 253
3.1-4 . . . 256
3.3-1 . . . 258
3.3-2 . . . 259
3.3-3 . . . 260
3.3-4 . . . 262
3.3-5 . . . 263
A.2 Zdroje . . . 265
A.3 Index prvkov SQL . . . 266
A.4 Index pojmov . . . 267

0
Úvod

0.1 História databázových technológií

Prvé počítačové spracovanie údajov sa datuje do 50. rokov. Za ten čas vzniklo niekoľko databázových modelov, každý z nich, samozrejme, priniesol mnoho zaujímavých teoretických i praktických noviniek, a ovplyvnil tak ďalší vývoj.

Pri najstaršom databázovom modeli využívajúcom **správu súborov** sa jednotlivé dáta ukladajú sekvenčne do jedného veľkého súboru (na disku alebo na magnetickej páske či dierovom štítku). Hlavnými nevýhodami tejto metódy sú predovšetkým nutná znalosť spôsobu a miesta fyzického uloženia údajov a priveľká časová náročnosť prístupu k dátam, preto sa v súčasnosti v komerčných produktoch vôbec nevyužíva.

Základom **hierarchického** modelu je stromová štruktúra, v ktorej môžu byť uzly prepojené (prostredníctvom smerníkov) v klasickom hierarchickom vzťahu rodič-dieťa, ale aj v rámci jednej úrovne. Pomocou navigácie po vetvách možno nájsť požadovanú informáciu. Oproti predchádzajúcemu modelu je popri zväčšenej efektivite vyhľadávania hlavnou výhodou odclonenie fyzickej implementácie. Medzi hlavné obmedzenia hierarchického modelu patrí v prípade zmeny požiadaviek nutnosť prepracovať celé štruktúry databázy. Vzhľadom na jednosmernosť vzťahu rodič-dieťa (hovoríme o väzbe typu 1:n) je problematická aj realizácia komplikovanejších väzieb (typu m:n). Najznámejšou implementáciou hierarchického modelu bol systém IMS vyvinutý spoločnosťami IBM a NAA v rámci kozmického projektu Apollo v 60. rokoch na organizovanie dvoch miliónov súčiastok. Tento databázový model je (popri modernejších metódach) možné použiť ešte aj dnes, a to predovšetkým pri programoch spracovávajúcich dáta založené na hierarchickej štruktúre, ako napríklad organizačné či skladové systémy.

Počiatky **sieťového** modelu spadajú do polovice 60. rokov, no prvej konkrétnej špecifikácie sa dočkal až v apríli 1971, keď ho popísal výbor Database Task Group v rámci konferencie CODASYL (Conference on Data System Languages). Jeho podstatou je použitie smerníkov vyjadrujúcich vzťahy medzi jednotlivými databázovými položkami, čím vzniká sieťová štruktúra znázorňovaná na tzv. Bachmannovom diagrame. Nepodarilo sa odstrániť obmedzenia pri zmene štruktúry, no oproti hierarchickému modelu tu možno pomerne jednoducho realizovať väzby typu m:n. I keď mal kedysi tento model mnohé rozšírenia, od polovice 70. rokov bol postupne plne nahradzovaný relačným databázovým modelom.

V roku 1970 publikoval **Edgar Frank Codd** článok **A Relational Model of Data for Large Shared Data Banks**, v ktorom položil matematické základy dodnes najrozšírenejšieho databázového modelu – **relačného**. Uvedomil si, že všetky údaje v databáze sa dajú reprezentovať jednotným spôsobom ako hodnoty v tabuľkách, ktoré možno vyjadriť matematickým pojmom relácie, čím získal silný formálny nástroj na popis celej štruktúry. Veľkým prínosom relačného modelu je tiež fakt, že kladie dôraz na zachovanie integrity spracovávaných dát. Na tomto modeli je dnes založená drvivá väčšina komerčných databázových systémov (IBM **DB2**, **Oracle** (v súčasnosti majú oba zhruba tretinový podiel na databázovom trhu), **MS SQL Server** (asi šestinu), **Sybase**, **Informix**, **PostgreSQL**, **Ingres**, **MySQL**, **MS Access**, **FoxPro**, ...).

Moderný **objektový** databázový model podporuje väčšinu prvkov z objektového prístupu – zložené objekty či dedičnosť. Jednotlivé záznamy už nie sú tvorené len samotnými hodnotami, ako je to v relačnom modeli, ale obsahujú aj vlastné metódy, ktoré s týmito hodnotami pracujú. Vzhľadom na dlhoročne osvedčenú výkonnosť a bezpečnosť produktov, ktoré využívajú relačný databázový model, však nemožno očakávať, že relačný model bude plne nahradený objektovým, skôr pôjde o vzájomné zbližovanie oboch prístupov.

Veľkému záujmu sa i naďalej budú tešiť špecializované modely navrhnuté na spracovanie neštruktúrovaných a semištruktúrovaných informácií (tu patrí predovšetkým bežný text a populárny XML formát), a to tzv. dokumentografické systémy či vektorový dátový model.

Svet je čoraz rýchlejší a množstvo informácií nesmierne narastá. Ako pars pro toto uveďme, že jedna z najambicióznejších súčasných databázových úloh je súčasťou projektu urýchľovača Large Hadron Collider v švajčiarskom CERN. Ide o vytvorenie distribuovanej databázy schopnej spravovať jeden exabyte (t. j. 10^{18} bytov) údajov. Je preto nutné aj naďalej zdokonaľovať metódy ukladania dát, aby boli rýchlo prístupné. To vysvetľuje trvalý záujem o problematiku databázových systémov.

0.2 História SQL

Coddov článok osnovajúci relačný databázový model podnietil rozsiahle výskumy. V rokoch 1974 až 1979 firma IBM, ktorá Codda zamestnávala, vo svojich výskumných laboratóriách v San José vypracovala prototyp relačnej databázy zvanej System/R, ktorý zverejnila v roku 1978. Ním celému informatickému svetu prvýkrát ukázala schodnosť Coddovej teórie. V rámci tohto projektu D. Chamberlin v roku 1974 definoval jazyk, ktorý podporoval prístup viacerých používateľov a dopyty na dáta vo viacerých tabuľkách. Pôvodný (trochu megalomanský) názov Structured English Query Language (so skratkou SEQUEL) bol neskôr zjednodušený na **Structured Query Language** – **SQL**.

Skupina inžinierov sledujúca projekt System/R pochopila potenciál relačných databáz. Založila spoločnosť Relational Software, Inc. a v roku 1979 vytvorila prvý komerčne dostupný relačný databázový systém Oracle založený na dopytovacom jazyku SQL. Druhým bol Ingres, vypracovaný spoločnosťou Relational Technology, ten však mal svoj vlastný dopytovací jazyk QUEL (v roku 1986 sa aj Ingres preorientoval na používanie SQL). Nezaháľala ani firma IBM – zhruba v roku 1983 vyvinula ďalšie vlastné produkty SQL/Data System (SQL/DS), QMF a Database 2 (DB2). Vzhľadom na postavenie tejto spoločnosti na trhu sa jej verzia SQL stala de facto štandardom.

Odvtedy sa na trhu objavilo mnoho databázových systémov a všetky podporovali SQL ako svoj primárny jazyk. S masovým rozšírením SQL však rástla pravdepodobnosť nedorozumení, preto v rokoch 1986 a 1987 Americký národný normalizačný ústav (American National Standards Institute, **ANSI**) a Medzinárodná normalizačná organizácia (International Standards Organization, **ISO**) vypracovali definíciu štandardu pod názvom SQL86. Stal sa ním (už aj de iure) dialekt SQL firmy IBM – bol charakterizovaný ako „prienik existujúcich implementácií“. (Aj to je jeden z dôvodov, prečo sa ďalej budeme venovať práve produktu tejto spoločnosti pod (zjednodušeným) názvom DB2.)

Od tých čias zverejnila ANSI-SQL skupina tri ďalšie štandardy: **SQL89** (obsahuje už integritné obmedzenia), **SQL92** (podporuje integráciu s niektorými programovacími jazykmi) a **SQL99** (rozširuje doterajší stav hlavne v oblasti internetovského prístupu k dátam (špeciálne s dôrazom na XML štandardy) a integrácie s jazykom Java (najmä prostredníctvom Java Database Connectivity (JDBC))). Podstatné črty SQL však (na rozdiel od iných oblastí informatiky) ostávajú nezmenené, čo teší hlavne priaznivcov spätnej kompatibility. Svedčia o tom aj pomerne veľké časové odstupy medzi definíciami SQL92, SQL99 a súčasnosťou.

Rôzne databázové systémy (tak ako každý iný tovar) majú rôznu kvalitu, ktorá sa, samozrejme, odráža v ich cene. Aj tu záleží len na rozhodnutí zákazníka, či mu na ním zamýšľanú databázu postačí slabší a lacnejší produkt, alebo či naozaj dokáže využiť kvalitu drahšieho. Väčšina slušných databázových systémov dnes (aspoň v podstatných rysoch) spĺňa štandard SQL92.

Tak ako iné odvetvia priemyslu, aj počítačový zo štandardov na jednej strane ťaží, a na strane druhej nimi trpí. **SQL je otvorený štandardný jazyk, ktorý nie je vlastníctvom žiadnej spoločnosti.** Preto je jeho normalizovaná verzia od ANSI považovaná za „čisté“ SQL. Niektoré databázové spoločnosti sa však cítia týmto štandardom príliš zväzované. Do svojich SQL preto pridávajú nové črty, ktoré nie sú v ANSI-SQL popísané, čím vznikajú neprenosné **dialekty SQL**. To však neznamená, že tento fenomén treba zavrhnúť, veď vo všeobecnosti práve **napätie medzi potrebou existencie noriem a potrebou ich porušovania je významným zdrojom pokroku**. (Napríklad ANSI-SQL neobsahuje automatické očíslovanie nových záznamov, ale väčšina dnes predávaných databázových systémov už túto (u programátorov obľúbenú) možnosť podporuje (bohužiaľ, DB2 zatiaľ nie). Dá sa preto očakávať, že táto črta sa vyskytne v nasledujúcom štandarde SQL.)

0.3 Princípy databázových systémov a SQL

Databázová technológia je unifikovaný súbor pojmov, prostriedkov a techník slúžiaci na vytváranie informačných systémov. Údaje sú organizované v databáze a sú riadené balíkom programov úzko spolupracujúcich s príslušným operačným systémom. Obe tieto zložky – **databáza** a **systém jej riadenia** – tak vytvárajú **databázový systém**, ktorý možno považovať za **jadro informačného systému**.

Ako už vieme, databázové systémy môžu byť založené na rôznych modeloch, a tak vychádzať z rôznych princípov. Avšak bez ohľadu na ne musia byť údaje v každom databázovom systéme nevyhnutne **perzistentné** (t. j. musia v ňom pretrvávať aj vtedy, keď s nimi nepracuje žiaden program alebo iný používateľ) a **spoľahlivé** (čiže modifikované môžu byť len na výzvu (oprávneného) používateľa).

Okrem týchto úplne prirodzených a skôr technických podmienok však každý databázový systém hoden svojho mena musí spĺňať ešte jeden dôležitý princíp – **neduplicitu** dát, teda zásadu, aby **sa totožná informácia v žiadnom prípade neopakovala na viacerých miestach**.

Databázový systém by mal dokonca rešpektovať ešte silnejšiu požiadavku – **neredundantnosť** dát, keď nie sú prípustné ani údaje, ktoré možno odvodiť z iných. Táto vlastnosť je však občas v rozpore s ďalším princípom – **rýchlosťou prístupu** k dátam, takže tvorca databázového modelu by mal pri takomto konflikte dobre zvážiť, ktorú z týchto zásad poruší. Cenou za rýchly prístup je totiž nákladnejšia starostlivosť o konzistenciu prvotných a odvodených dát. Pre databázu to znamená použiť zložitejšie prvky (tzv. triggery), v opačnom prípade by táto úloha ostávala na často nedisciplinovaných klientských programoch.

SQL je založené na relačnom modeli dát. Dáta sú používateľovi prezentované vždy vo forme **tabuľky** – obdĺžnikovej matice s riadkami a stĺpcami. Tabuľky a ich stĺpce majú svoje jednoznačné mená, pomocou ktorých k nim možno pristupovať.

Definície charakteru dát v takejto databáze – tzv. **metadát** alebo **logickej schémy** databázy – sa formulujú v špeciálnom jazyku, ktorý okrem tabuliek a stĺpcov umožňuje definovať aj niektoré vzťahy medzi údajmi – tzv. **integritné obmedzenia**. O manipuláciu s jednotlivými dátami sa stará ďalší jazyk, v ktorom sa dajú formulovať požiadavky na vkladanie, odstraňovanie a modifikáciu údajov. Obsahuje tiež prostriedky umožňujúce klásť jednoduchý i náročnejší **dopyt** (často sa používa aj anglické slovo **query** (a, žiaľ, nesprávne aj české slovo „**dotaz**“)) – otázku na údaje v databáze. Takýto jazyk preto nazývame **dopytovací**.

SQL je jazykom, ktorý zahŕňa obe tieto funkcie – je v ňom možné zadefinovať metadáta a potom pracovať aj s jednotlivými dátami. Na rozdiel od klasických **procedurálnych** programovacích jazykov typu Pascal, kde program spočíva v popísaní postupu (algoritmu), je SQL tzv. **deklaratívnym** jazykom – programátor len kladie požiadavku, spravidla ho nezaujíma, ako sa tento príkaz realizuje. O jeho preklad a vykonanie sa už postará systém riadenia databázy. Kvôli jednoduchej kontrole sa riadi zásadou, že **príkaz sa buď vykoná celý, alebo sa nevykoná vôbec**.

SQL môže existovať samostatne, no býva aj súčasťou rozšírenia iných programovacích jazykov.

1
Práca s jednou tabuľkou

1.1 Každý začiatok je – ľahký

1.1.1 Prieskum terénu

Predstavme si, že máme s kolegami za úlohu vytvoriť informačný systém o študentoch istej vysokej školy, ktorý by mal obsahovať ich mená a priezviská, dátumy narodenia, ročník a študijné priemery. Hneď sa s chuťou pustíme do práce, no, ako to už býva, okamžite zistíme, že nemáme k dispozícii žiadne údaje, a to ani len ich štruktúru. Povieme si možno, že zháňať dáta nie je naša starosť, no zadávateľ nás rýchlo presvedčí, že on nechce žiadne naše pekné teórie, ale funkčný systém. Pod hrozbou finančnej straty ľahko pochopíme, že údaje si musíme niekde získať sami. Na tamojšom študijnom oddelení nám najprv odmietnu poskytnúť informácie, lebo nemáme povolenie od dekana. Po týždni, keď sa dekan vráti zo služobnej cesty, vysvitne, že také potvrdenie sme mohli získať aj od jeho prodekana pre štúdium, i keď ten tvrdil opak. S papierom bežíme na študijné oddelenie, ale tu nám po hodinovom prehľadávaní skrine so zle označenými fasciklami poskytnú len zažltnutý zoznam študentov so skratkami krstných mien, aj to s poznámkou, že odvtedy určite nejakí študenti pribudli i odišli. Navyše, zoznam neobsahuje dátumy narodenia, lebo tie boli podľa toho a toho zákona platného v čase vyhotovenia zoznamu považované za dôverné. Ročník sa, samozrejme, u väčšiny študentov zmenil. Keď požiadame o študijné výsledky, odmietnu nám ich vydať, lebo o tom hovorí nejaká interná smernica. Po intervencii známeho, ktorého brat je šéfom syna panej zo študijného, nám táto pani radšej študijné výsledky vydá, no vysvitne, že v nich ešte nie je zohľadnené minuloročné hodnotenie. Obetujeme bonboniéru, a získame tak možnosť prehľadať príslušné spisy. Ukáže sa, že jeden študent má tri dátumy narodenia a inému, ktorého niekto z nás pozná už od detstva, v mene chýba dĺžeň. Popri tejto práci sme nútení pochopiť, že študijné výsledky nevzniknú vždy ako jednoduchý aritmetický priemer, a musíme si naštudovať spôsob, ako sa to v takýchto výnimočných situáciách vlastne počíta... Treba pokračovať?

Takto sme zakúsili azda najdôležitejšiu a najzaujímavejšiu časť práce analytika – „prieskum terénu“. **Ten by mal viesť k dôkladnému pochopeniu doteraz možno úplne neznámej problematiky, a to dokonca do takej miery, aby ju bolo možné formalizovať.** Chceli sme síce utvoriť informačný systém, ale popritom sme si urobili základný kurz práva či ekonómie, získali niekoľko zaujímavých známych, a aj aký-taký prehľad o (ne)fungovaní nášho školstva a štátu vôbec. Napokon, veď práve **narastajúce a čoraz nezvládnuteľnejšie množstvo dát je ten najvážnejší dôvod na existenciu informačného systému**, a tak nepochybne ono bolo pravou príčinou, prečo nás oň zadávateľ požiadal.

1.1.2 Vytvorenie tabuľky

Po niekoľkotýždňovej krvopotnej práci sa nám povedzme podarilo získať takéto informácie (kvôli jednoduchosti predpokladajme, že študentov je iba 13):

- Ján Hraško: dátum nar. 12. 7. 1987, prvák, priemer 1,83
- Ružena Šípová: dátum 1. 2. 1984, priem. 1,22, 1. roč.
- Aladár Baba: druhák, nar. 22. jan. 1980, š. v. 2,03
- Ferdinand Mravec: 3. 3. 1984, 3. r., čistý jednotkár, cena rektora
- Polienko Ján: ročník 5, dátum 14. 4. 1982, známky 2,28
- Juraj Truľo: dátum narodenia: 16. 7. 1979, študijné výsledky: 3, ročník: 1.
- Jana Botková: 21. 9. 1977, 4. ročník, zn. 1,5
- Dana $-||-$, okrem známky – 1,4
- Ján Hlúpy: nezistený, druhák, čistý trojkár
- Miazga Aladár: nar. 22. decembra 1987, 3. r., zn. 2,06
- Mikuláš Myšiak: date of birth June 6th, 1983, 5th year, average 1.66, predtým študoval na Oxforde
- Donald Káčer: date of birth October 7th, 1982, 5th year, average 1.83, predtým študoval na Oxforde

- Jozef Námorník: d. n. 23. 9. 1981, obč. Kanada, r. 2, p. 2,90

Aj keď majú údaje v jednotlivých riadkoch rôznu formu (nečudo, veď ich zbierali rôzni ľudia), cítime, že vypovedajú o skutočnostiach rovnakého druhu. Napríklad „12. 7. 1987“, „22. decembra 1987“, či „June 6th, 1983“ sú dátumy (z kontextu vieme, že dátumy narodenia) a „prvák“, „3. r.“, či „1. roč.“ reprezentujú celé číslo (sú to predsa čísla ročníkov). Hovoríme, že sú rovnakého **dátového typu**. Netreba veľkú dávku systematického myslenia, aby si človek uvedomil, že tieto údaje bude prehľadnejšie zapísať do tabuľky. Tak lepšie ustriehneme aj to, aby údaje rovnakého typu mali i rovnakú formu. Popritom sa pozbavujeme irelevantných informácií (napr. „cena rektora“), vymeníme prípadné nesprávne poradie položiek (napr. Ján Polienko má prehodené meno a priezvisko), rozšifrujeme všemožné skratky a pridáme i niektoré implicitné údaje, ktoré vyplývajú z kultúrneho kontextu a tradícií (napr. informáciu o pohlaví – uvedomme si, že trebárs študentovi Námorníkovi prisúdime mužské pohlavie len preto, že doteraz každý Jozef, s ktorým sme sa stretli, bol muž). Dostávame takúto tabuľku:

meno	priezvisko	pohlavie	dátum narodenia	ročník	priemer
Ján	Hraško	muž	12. 7. 1987	1	1,83
Ružena	Šípová	žena	1. 2. 1984	1	1,22
Aladár	Baba	muž	22. 1. 1980	2	2,03
Ferdinand	Mravec	muž	3. 3. 1984	3	1,00
Ján	Polienko	muž	14. 4. 1982	5	2,28
Juraj	Truľo	muž	16. 7. 1979	1	3,00
Jana	Botková	žena	21. 9. 1977	4	1,50
Dana	Botková	žena	21. 9. 1977	4	1,40
Ján	Hlúpy	muž	nezistený	2	3,00
Aladár	Miazga	muž	22. 12. 1987	3	2,06
Mikuláš	Myšiak	muž	6. 6. 1983	5	1,66
Donald	Káčer	muž	7. 10. 1982	5	1,83
Jozef	Námorník	muž	23. 9. 1981	2	2,90

To, čo sme robili doteraz, nie je závislé od databázového systému – takúto tabuľku si predsa pokojne môžeme nakresliť aj na papier. Na to, aby sme mohli využívať služby databázy, musíme najprv údaje z tejto tabuľky do nej dostať. Budeme to robiť v dvoch prirodzených krokoch – najprv zostrojíme prázdnu tabuľku a potom ju naplníme dátami.

Pripomeňme ešte raz, že sa zaoberáme DB2 dialektom SQL. Na skonštruovanie prázdnej tabuľky použijeme takýto jeho príkaz:

```
CREATE TABLE študent
(
  meno            VARCHAR(10),
  priezvisko      VARCHAR(15),
  pohlavie        CHAR(4),
  dátum_narodenia DATE,
  ročník          INT,
  priemer         DEC(3,2)
)
```

Všimnime si bližšie jeho syntax. Reťazec `CREATE TABLE` je anglickým prekladom (a tak to obvykle bude aj naďalej) slovného spojenia „vytvor tabuľku“. Nasleduje meno tabuľky `študent`, ktoré sme si zvolili sami, a nie je teda súčasťou SQL. V okrúhlych zátvorkách sú potom vymenované stĺpce oddelené čiarkami. Každý z nich má svoje meno (ktoré sme opäť zvolili my) a dátový typ. Sú tu zastúpené všetky základné dátové typy:

- `VARCHAR(`x`)` (alebo ekvivalentne `CHARACTER VARYING(`x`)` či `CHAR VARYING(`x`)`) slúži na ukladanie ľubovoľných reťazcov znakov (tzv. stringov) dĺžky najviac x (pričom $x \leq 2^{16} - 2$).

- CHAR(x) (alebo v neskrátenej podobe CHARACTER(x)) sa tiež používa na ukladanie ľubovoľných reťazcov, tu však majú pevnú dĺžku x (tu je jeho hranica omnoho menšia, a to $2^8 - 2$, čo je spôsobené inou implementáciou). Dodajme, že ak sú ukladané stringy kratšie, jednoducho sa automaticky doplnia príslušným počtom medzier.
- INT (alebo neskrátene INTEGER) slúži na ukladanie celých čísel (samozrejme, v určitom rozsahu, konkrétne od -2^{31} do $2^{31} - 1$).
- Do typu DEC(x,y) (alebo aj DECIMAL(x,y), ale ekvivalentne aj NUM(x,y) či NUMERIC(x,y)) možno uložiť desatinné čísla s celkovou dĺžkou (t. j. počtom cifier pred i za desatinnou čiarkou či bodkou) $x \leq 31$ a počtom desatinných miest $y \leq x$.
- DATE sa používa na ukladanie dátumov (s najviac štvorciferným rokom, čo však bohato stačí).

Popri nich treba spomenúť existenciu dátových typov na čas (TIME), „dátumočas“ (alebo tzv. „časovú pečiatku“) (TIMESTAMP), ďalších celočíselných a desatinnočíselných typov, líšiacich sa rozsahom (SMALLINT a BIGINT, resp. REAL či DOUBLE). Ak potrebujeme uložiť extrémne veľké reťazce alebo binárne objekty, možno použiť špeciálne dátové typy CLOB a BLOB.

Názvy stĺpcov ani tabuliek spravidla neobsahujú medzery, v prípade potreby ich možno nahradiť podčiarkovníkmi (_) (ako je to v prípade stĺpca `dátum_narodenia`). Ak však na medzerách trváme, musíme príslušný názov písať v úvodzovkách (") (v našom prípade by to bolo `"dátum narodenia"`).

Postrehli sme už, že nami volené názvy tabuliek či stĺpcov môžu obsahovať aj diakritické znamienka. Vzhľadom na možné technické problémy je však praktické vyhýbať sa im (napriek tomu ich tu však tvrdohlavo používať budeme). Z pochopiteľných dôvodov je ohraničená aj dĺžka týchto názvov (väčšinou ide o hranicu 18).

Tu i ďalej dodržiavame takúto dohodu o veľkosti písmen v príkaze: Veľkými písmenami budeme písať tie časti, ktoré sú súčasťou jazyka SQL (v našom prípade napr. `CREATE TABLE` alebo dátové typy `VARCHAR`, `DATE`, `DEC`, `INT`). Malými písmenami naznačíme, že nejde o súčasť jazyka, ale o nami zvolené názvy (v tomto prípade napr. názov tabuľky `študent` alebo názvy stĺpcov `meno`, `priezvisko`, `dátum_narodenia`, `ročník` a `priemer`). Treba však povedať, že SQL (aspoň vo verzii DB2) veľkosť písmen v príkazoch nerozlišuje (pravdaže, okrem reťazcových hodnôt, kde na veľkosti naopak záleží).

V tomto okamihu máme v databáze vytvorenú prázdnu tabuľku. Nasleduje druhá fáza – jej naplnenie. Opäť poslúži SQL, a to takýmto príkazom (všimnime si pri tom, že nezistený dátum narodenia u Jána Hlúpeho sa premietol do **prázdnej hodnoty**, ktorú budeme označovať NULL):

```
INSERT INTO študent
VALUES
  ('Ján',       'Hraško',   'muž',  '12.7.1987',  1, 1.83),
  ('Ružena',    'Šípová',   'žena', '1.2.1984',   1, 1.22),
  ('Aladár',    'Baba',     'muž',  '22.1.1980',  2, 2.03),
  ('Ferdinand', 'Mravec',   'muž',  '3.3.1984',   3, 1.00),
  ('Ján',       'Polienko', 'muž',  '14.4.1982',  5, 2.28),
  ('Juraj',     'Truľo',    'muž',  '16.7.1979',  1, 3.00),
  ('Jana',      'Botková',  'žena', '21.9.1977',  4, 1.50),
  ('Dana',      'Botková',  'žena', '21.9.1977',  4, 1.40),
  ('Ján',       'Hlúpy',    'muž',  NULL,         2, 3.00),
  ('Aladár',    'Miazga',   'muž',  '22.12.1987', 3, 2.06),
  ('Mikuláš',   'Myšiak',   'muž',  '6.6.1983',   5, 1.66),
  ('Donald',    'Káčer',    'muž',  '7.10.1982',  5, 1.83),
  ('Jozef',     'Námorník', 'muž',  '23.9.1981',  2, 2.90)
```

Opäť pár slov o syntaxi. Klauzula INSERT znamená „vlož“, spojka INTO je „do“, `študent` je meno tabuľky, do ktorej ideme ukladať dáta, a `VALUES` znamená „hodnoty“. Dokopy teda prikazujeme „vlož do (tabuľky) `študent` hodnoty“. A nasledujú hodnoty – a to celé riadky, tzv. **záznamy**. Každý z nich je uzavretý v okrúhlych zátvorkách a navzájom sú oddelené čiarkami. Jednotlivé údaje v zázname – tzv. **položky** – sú navzájom oddelené čiarkami. V každom zázname je presne toľko položiek, koľko stĺpcov má tabuľka. Ich poradie určuje, ktorá bude uložená do ktorého stĺpca (napr. pri zázname `('Ján','Hraško','muž','21.7.1987',1,1.83)` bude položka `Ján` uložená do stĺpca `meno`, `Hraško` do stĺpca `priezvisko`, `muž` do `pohlavie`, `12.7.1987` do `dátum_narodenia`, `1` do `ročník` a napokon `1.83` (všimnime si desatinnú bodku) do stĺpca `priemer`).

Z toho vyplýva, že položka a dátový typ stĺpca musia navzájom korešpondovať – nemôžeme napríklad hodnotu `12.7.1987` vložiť do stĺpca `ročník`, ktorý je typu `INT`, databázový systém by zahlásil chybu a príkaz by nevykonal. (Naproti tomu by sa nám podarilo napríklad túto hodnotu, ktorá sa dá považovať aj za reťazec, vložiť do stĺpca `priezvisko` typu `VARCHAR(15)` – pamätajme na to, že **systém je automat a nerozumie obsahu vkladaných údajov**.) S dátovými typmi súvisí ešte jedna dohoda – číselné údaje (napr. `INT` či `DEC`) ani prázdnu hodnotu nepíšeme do apostrofov (`'`), kým všetky ostatné áno.

1.2 Výpisy všetkých záznamov

1.2.1 Základný výpis

No dobre, dáta sme síce do databázy dostali, ale vieme sa k nim dostať? Samozrejme, že áno. Základným príkazom v jazyku SQL, ktorým vieme prečítať obsah našej tabuľky `študent`, je takáto konštrukcia:

```
SELECT *
FROM študent
```

Klauzula SELECT znamená „vyber“ a FROM „z“. Hviezdičku * môžeme čítať „všetko“, takže celkovo tento príkaz hovorí „vyber všetko z (tabuľky) `študent`“ (pravdaže, pod vybraním rozumieme len prečítanie, v skutočnosti dáta v tabuľke ostávajú). Takýto príkaz nazývame **dopyt** (a ešte častejšie slangovo „**selekt**“). Ako **odpoveď** naň v tomto prípade dostávame:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ružena	Šípová	žena	1984-02-01	1	1,22
Aladár	Baba	muž	1980-01-22	2	2,03
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Ján	Polienko	muž	1982-04-14	5	2,28
Juraj	Truľo	muž	1979-07-16	1	3,00
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40
Ján	Hlúpy	muž	NULL	2	3,00
Aladár	Miazga	muž	1987-12-22	3	2,06
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Jozef	Námorník	muž	1981-09-23	2	2,90

Vidíme, že táto odpoveď (ako v budúcnosti každá ďalšia) má formu tabuľky. V jej záhlaví sú mená stĺpcov, tak ako sme ich uviedli v definícii tabuľky, potom už nasledujú jednotlivé záznamy. (Všimnime si ešte technické detaily – zmenu veľkosti písmen v názvoch stĺpcov (lebo neboli definované s úvodzovkami), inú formu údajov typu DATE a desatinnú čiarku namiesto bodky, je to však len záležitosť výpisu.)

Prejdime k prvej „pridanej hodnote“ použitia databázy – schopnosti zoradiť záznamy podľa hodnôt ľubovoľného stĺpca (ba dokonca aj podľa viacerých). Pri použití predošlého príkazu sú záznamy vypísané nezoradene (spravidla v poradí, v akom boli do tabuľky vkladané). Ak chceme študentov zoradiť podľa hodnôt stĺpca `priezvisko`, príkaz doplníme na takýto tvar:

```
SELECT *
FROM študent
ORDER BY priezvisko
```

alebo (ekvivalentne):

```
SELECT *
FROM študent
ORDER BY 2
```

Klauzula ORDER BY znamená doslova „poradie podľa“, nasleduje meno príslušného stĺpca alebo jeho poradové číslo. Odpoveď je v oboch prípadoch rovnaká:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Aladár	Baba	muž	1980-01-22	2	2,03
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40
Ján	Hlúpy	muž	NULL	2	3,00
Ján	Hraško	muž	1987-07-12	1	1,83
Donald	Káčer	muž	1982-10-07	5	1,83
Aladár	Miazga	muž	1987-12-22	3	2,06
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Jozef	Námorník	muž	1981-09-23	2	2,90
Ján	Polienko	muž	1982-04-14	5	2,28
Ružena	Šípová	žena	1984-02-01	1	1,22
Juraj	Truľo	muž	1979-07-16	1	3,00

Vidíme, že ku klasickému usporiadaniu podľa abecedy predsa len čosi chýba – i keď sú Janka a Danka Botkové podobné ako vajce vajcu, predsa by sa mali vymeniť. Máme síce usporiadanie podľa priezviska ako doteraz, ale týmto stĺpcom nerozlíšené záznamy ešte potrebujeme zoradiť podľa hodnôt stĺpca meno. V SQL príkaze sa to prejaví takto:

```
SELECT *
FROM student
ORDER BY priezvisko, meno
```

alebo:

```
SELECT *
FROM student
ORDER BY 2, 1
```

ale tiež (priznajme, že nie veľmi konzistentne):

```
SELECT *
FROM student
ORDER BY 2, meno
```

Za klauzulou ORDER BY teda môže byť viacero stĺpcov, resp. ich poradových čísel, ktoré sú potom oddelené čiarkami. Odpoveď je vo všetkých prípadoch podľa očakávania totožná:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Aladár	Baba	muž	1980-01-22	2	2,03
Dana	Botková	žena	1977-09-21	4	1,40
Jana	Botková	žena	1977-09-21	4	1,50
Ján	Hlúpy	muž	NULL	2	3,00
Ján	Hraško	muž	1987-07-12	1	1,83
Donald	Káčer	muž	1982-10-07	5	1,83
Aladár	Miazga	muž	1987-12-22	3	2,06
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Jozef	Námorník	muž	1981-09-23	2	2,90
Ján	Polienko	muž	1982-04-14	5	2,28
Ružena	Šípová	žena	1984-02-01	1	1,22
Juraj	Truľo	muž	1979-07-16	1	3,00

Vedeli by sme zoradiť študentov od najstaršieho po najmladšieho? Isteže, táto informácia je skrytá v 4. stĺpci – dátum_narodenia, takže napríklad takto:

```
SELECT *
FROM student
ORDER BY 4
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40
Juraj	Truľo	muž	1979-07-16	1	3,00
Aladár	Baba	muž	1980-01-22	2	2,03
Jozef	Námorník	muž	1981-09-23	2	2,90
Ján	Polienko	muž	1982-04-14	5	2,28
Donald	Káčer	muž	1982-10-07	5	1,83
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Ružena	Šípová	žena	1984-02-01	1	1,22
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Ján	Hraško	muž	1987-07-12	1	1,83
Aladár	Miazga	muž	1987-12-22	3	2,06
Ján	Hlúpy	muž	NULL	2	3,00

Všimnime si, že Ján Hlúpy s neznámym dátumom narodenia je na chvoste, keďže prázdna hodnota sa pri usporadúvaní vždy správa, akoby bola najväčšia.

No dobre, ale čo keď potrebujeme opačné, zostupné poradie, od najmladšieho k najstaršiemu? V takom prípade v klauzule ORDER BY bezprostredne za príslušný stĺpec, resp. jeho číslo doplníme DESC, čo je skratka anglického slova „descending“ čiže „zostupne“. Takže:

```
SELECT *
FROM student
ORDER BY 4 DESC
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hlúpy	muž	NULL	2	3,00
Aladár	Miazga	muž	1987-12-22	3	2,06
Ján	Hraško	muž	1987-07-12	1	1,83
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Ružena	Šípová	žena	1984-02-01	1	1,22
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Ján	Polienko	muž	1982-04-14	5	2,28
Jozef	Námorník	muž	1981-09-23	2	2,90
Aladár	Baba	muž	1980-01-22	2	2,03
Juraj	Truľo	muž	1979-07-16	1	3,00
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40

Tentoraz je Ján Hlúpy na začiatku.

Dodajme, že na vzostupné usporiadanie možno analogicky použiť aj ASC (skratka anglického slova „ascending“ čiže „vzostupne“), nemá to však žiaden praktický význam. Napríklad dopyty:

```
SELECT *
FROM student
ORDER BY 5 DESC, 6 ASC
```

a:

```
SELECT *
FROM student
ORDER BY 5 DESC, 6
```

dávajú rovnakú odpoveď – študenti sú usporiadaní podľa ročníkov zostupne a v rámci ročníkov podľa priemeru vzostupne:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Ján	Polienko	muž	1982-04-14	5	2,28
Dana	Botková	žena	1977-09-21	4	1,40
Jana	Botková	žena	1977-09-21	4	1,50
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Aladár	Miazga	muž	1987-12-22	3	2,06
Aladár	Baba	muž	1980-01-22	2	2,03
Jozef	Námorník	muž	1981-09-23	2	2,90
Ján	Hlúpy	muž	NULL	2	3,00
Ružena	Šípová	žena	1984-02-01	1	1,22
Ján	Hraško	muž	1987-07-12	1	1,83
Juraj	Truľo	muž	1979-07-16	1	3,00

Ako bonus k tejto stati dodajme, že existuje ešte jeden spôsob, ako databázu donútiť k výpisu údajov vo forme tabuľky, a to vtedy, keď v nej nie sú uložené, ale zadané explicitne, priamo v príkaze. Nezačína sa slovom SELECT, ale VALUES („hodnoty“), za ktorým nasleduje jeden alebo viac (budúcich) riadkov. Pri tých musíme dávať pozor, aby obsahovali rovnaký počet položiek zodpovedajúcich typov a aby boli (v prípade, že majú aspoň dve položky) v zátvorkách a navzájom oddelené čiarkami.

Takéto hodnoty môžeme dokonca aj usporiadavať. Napríklad:

```
VALUES
  ('Homer',  'Simpson'),
  ('Marge',  'Simpson'),
  ('Bart',   'Simpson'),
  ('Lisa',   'Simpson'),
  ('Maggie', 'Simpson')
ORDER BY 1
```

dáva odpoveď:

Bart	Simpson
Homer	Simpson
Lisa	Simpson
Maggie	Simpson
Marge	Simpson

V takejto nahej podobe síce vyzerá príkaz VALUES pomerne zbytočne, ale ešte príde čas, keď ho oceníme. Napokon, v istej forme sme sa s ním už stretli, keď sme do našej tabuľky vkladali dáta, bol časťou príslušného príkazu INSERT.

1.2.2 Iba niektoré stĺpce

Zatiaľ sme vždy vypisovali všetky dáta obsiahnuté v tabuľke, variovali sme len spôsob ich výpisu. Niekedy sa však môže stať, že práve v tej chvíli nepotrebujeme o záznamoch úplne všetky údaje. Predstavme si napríklad, že chceme zoznam študentov s ich študijnými výsledkami, ale dátum ich narodenia či ročník sú momentálne nepodstatné. Nemusíme preto vyrábať novú tabuľku, veď všetky potrebné údaje máme v tej našej. Stačí len povedať, ktoré stĺpce treba vypísať, a to tak, že napíšeme ich zoznam namiesto hviezdičky hneď za slovo SELECT v takom poradí, aké potrebujeme. V nasledujúcom príklade si všimnime, že nemusíme dokonca ani dodržať pôvodné poradie stĺpcov – stĺpec priezvisko je tu pred stĺpcom meno:

```
SELECT
  priezvisko,
  meno,
  priemer
FROM študent
```

Odpoveď:

PRIEZVISKO	MENO	PRIEMER
Hraško	Ján	1,83
Šípová	Ružena	1,22
Baba	Aladár	2,03
Mravec	Ferdinand	1,00
Polienko	Ján	2,28
Truľo	Juraj	3,00
Botková	Jana	1,50
Botková	Dana	1,40
Hlúpy	Ján	3,00
Miazga	Aladár	2,06
Myšiak	Mikuláš	1,66
Káčer	Donald	1,83
Námorník	Jozef	2,90

Tieto údaje môžeme tiež, samozrejme, usporiadať, a to takým spôsobom ako minule – menami príslušných stĺpcov alebo ich poradím. Ukazuje sa tu však niečo, čo sme si predtým nemohli všimnúť, keďže stĺpce každej odpovede na dopyt začínajúci sa `SELECT *` sa zhodujú so stĺpcami pôvodnej tabuľky. V nasledujúcom dopyte vidno, že `1` v klauzule `ORDER BY` sa týka 1. stĺpca odpovede, nie tabuľky `študent` (tam by šlo o stĺpec `meno`):

```
SELECT
  priezvisko,
  meno,
  priemer
FROM študent
ORDER BY 1
```

Odpoveď:

PRIEZVISKO	MENO	PRIEMER
Baba	Aladár	2,03
Botková	Jana	1,50
Botková	Dana	1,40
Hlúpy	Ján	3,00
Hraško	Ján	1,83
Káčer	Donald	1,83
Miazga	Aladár	2,06
Mravec	Ferdinand	1,00
Myšiak	Mikuláš	1,66
Námorník	Jozef	2,90
Polienko	Ján	2,28
Šípová	Ružena	1,22
Truľo	Juraj	3,00

Usporadúvať možno dokonca aj podľa stĺpca, ktorý nie je medzi vymenovanými. Ak napríklad chceme zoradiť študentov podľa veku od najmladšieho, pričom nás konkrétne dátumy ich narodení nezaujímajú, použijeme takýto dopyt:

```
SELECT
  priezvisko,
  meno,
  priemer
FROM študent
ORDER BY dátum_narodenia DESC
```

Odpoveď:

PRIEZVISKO	MENO	PRIEMER
Hlúpy	Ján	3,00
Miazga	Aladár	2,06
Hraško	Ján	1,83
Mravec	Ferdinand	1,00
Šípová	Ružena	1,22
Myšiak	Mikuláš	1,66
Káčer	Donald	1,83
Polienko	Ján	2,28
Námorník	Jozef	2,90
Baba	Aladár	2,03
Truľo	Juraj	3,00
Botková	Jana	1,50
Botková	Dana	1,40

Tu zrejme nemožno použiť alternatívnu verziu `ORDER BY` s poradovými číslami, pretože `dátum_narodenia` vo výslednej tabuľke nie je, a teda žiadne poradie nemá.

Vo výpise môžeme (ale nemusíme) každému stĺpcu nanútiť iný názov, tzv. **alias**. Napríklad `meno` premenujeme na `krstné_meno` takto:

```
SELECT
  priezvisko,
  meno AS krstné_meno,
  priemer
FROM študent
```

Slovíčko `AS` („ako“) slúžiace len na grafické oddelenie pôvodného názvu od aliasu môžeme pokojne vynechať:

```
SELECT
  priezvisko,
  meno krstné_meno,
  priemer
FROM študent
```

Odpoveď je v oboch prípadoch:

PRIEZVISKO	KRSTNÉ_MENO	PRIEMER
Hraško	Ján	1,83
Šípová	Ružena	1,22
Baba	Aladár	2,03
Mravec	Ferdinand	1,00
Polienko	Ján	2,28
Truľo	Juraj	3,00
Botková	Jana	1,50
Botková	Dana	1,40
Hlúpy	Ján	3,00
Miazga	Aladár	2,06
Myšiak	Mikuláš	1,66
Káčer	Donald	1,83
Námorník	Jozef	2,90

Aj alias môže obsahovať medzery, v tom prípade je však nutné ohraničiť ho úvodzovkami. Veľkosť písmen sa tu zachová:

```
SELECT
  priezvisko,
  meno AS "krstné meno",
  priemer
FROM študent
```

Odpoveď:

PRIEZVISKO	krstné meno	PRIEMER
Hraško	Ján	1,83
Šípová	Ružena	1,22
Baba	Aladár	2,03
Mravec	Ferdinand	1,00
Polienko	Ján	2,28
Truľo	Juraj	3,00
Botková	Jana	1,50
Botková	Dana	1,40
Hlúpy	Ján	3,00
Miazga	Aladár	2,06
Myšiak	Mikuláš	1,66
Káčer	Donald	1,83
Námorník	Jozef	2,90

Podľa aliasu možno aj usporadúvať:

```
SELECT
  priezvisko,
  meno AS "krstné meno",
  priemer
FROM študent
ORDER BY "krstné meno"
```

čo je ekvivalentné dopytu:

```
SELECT
  priezvisko,
  meno AS "krstné meno",
  priemer
FROM študent
ORDER BY meno
```

ale aj:

```
SELECT
  priezvisko,
  meno AS "krstné meno",
  priemer
FROM študent
ORDER BY 2
```

Odpoveď:

PRIEZVISKO	krstné meno	PRIEMER
Baba	Aladár	2,03
Miazga	Aladár	2,06
Botková	Dana	1,40
Káčer	Donald	1,83
Mravec	Ferdinand	1,00
Botková	Jana	1,50
Hraško	Ján	1,83
Polienko	Ján	2,28
Hlúpy	Ján	3,00
Námorník	Jozef	2,90
Truľo	Juraj	3,00
Myšiak	Mikuláš	1,66
Šípová	Ružena	1,22

Ukážme si v súvislosti s aliasmi ešte nejaké patologické situácie.

Aliasy rôznych stĺpcov môžu byť, ako ukazuje nasledujúci príklad, dokonca rovnaké:

```
SELECT
  priezvisko AS a,
```

```
  meno AS a,
  priemer
FROM študent
```

s odpoveďou:

A	A	PRIEMER
Hraško	Ján	1,83
Šípová	Ružena	1,22
Baba	Aladár	2,03
Mravec	Ferdinand	1,00
Polienko	Ján	2,28
Truľo	Juraj	3,00
Botková	Jana	1,50
Botková	Dana	1,40
Hlúpy	Ján	3,00
Miazga	Aladár	2,06
Myšiak	Mikuláš	1,66
Káčer	Donald	1,83
Námorník	Jozef	2,90

Usporiadanie podľa tohto aliasu:

```
SELECT
  priezvisko AS a,
  meno AS a,
  priemer
FROM študent
ORDER BY a
```

však zlyhá, t. j. dopyt nevygeneruje odpoveď, ale chybové hlásenie o duplicite názvov stĺpcov. To len ukazuje, že predchádzajúca tabuľka nebola úplne korektná.

Ani ďalšia tabuľka nie je (z rovnakého dôvodu) korektná, no tu sa, napodiv, usporadúvať podľa jedného z dvoch rovnomenných stĺpcov dá. Ukazuje to, že v takomto prípade má usporiadanie podľa pôvodného názvu väčšiu prioritu:

```
SELECT
  priezvisko AS meno,
  meno,
  priemer
FROM študent
ORDER BY meno
```

Odpoveď:

MENO	MENO	PRIEMER
Baba	Aladár	2,03
Miazga	Aladár	2,06
Botková	Dana	1,40
Káčer	Donald	1,83
Mravec	Ferdinand	1,00
Botková	Jana	1,50
Hraško	Ján	1,83
Polienko	Ján	2,28
Hlúpy	Ján	3,00
Námorník	Jozef	2,90
Truľo	Juraj	3,00
Myšiak	Mikuláš	1,66
Šípová	Ružena	1,22

A ešte jedna (tentoraz korektná) situácia, ktorá ukazuje, že pôvodný stĺpec po premenovaní túto svoju prioritu stráca:

```
SELECT
  priezvisko AS meno,
  meno AS krstné_meno,
  priemer
FROM študent
ORDER BY meno
```

Odpoveď:

MENO	KRSTNÉ_MENO	PRIEMER
Baba	Aladár	2,03
Botková	Jana	1,50
Botková	Dana	1,40
Hlúpy	Ján	3,00
Hraško	Ján	1,83
Káčer	Donald	1,83
Miazga	Aladár	2,06
Mravec	Ferdinand	1,00
Myšiak	Mikuláš	1,66
Námorník	Jozef	2,90
Polienko	Ján	2,28
Šípová	Ružena	1,22
Truľo	Juraj	3,00

Dodajme, že všetky doterajšie poznámky o usporiadaní možno aplikovať na každú tabuľku. Keďže ide iba o prezentáciu dát, klauzula ORDER BY sa (aj v budúcnosti) v každom dopyte môže vyskytnúť iba raz, a to vždy na jeho konci.

1.2.3 Stĺpcové funkcie

I keď databázový systém spravidla neslúži na formátovanie dát (o to sa zväčša starajú iné prostredia, ktoré z databázy len vytiahnu dáta v „surovom tvare“), ukážeme si niekoľko príkladov ilustrujúcich jeho možnosti.

Dosiaľ sme vypisovali stĺpce v podstate v takej forme, ako sú uložené v tabuľke. Teraz si ukážeme ďalšie možnosti výpisu, ktoré spočívajú v úpravách stĺpcov.

Všimnime si, že meno i priezvisko sú v samostatných stĺpcoch a sú medzi nimi v jednotlivých riadkoch nerovnako veľké medzery. Ak ich chceme obe dať do jediného stĺpca (a oddeliť jednou medzerou), potrebujeme ich „zlepiť“ – konkatenovať. Poslúži na to binárna funkcia CONCAT (z anglického „concatenate“ – „spojiť“) alebo ekvivalentne v infixnom tvare || (slangovo „**lepidlo**“), ktorá z dvoch reťazcov vytvorí jeden tak, že druhý pripojí za prvý:

```
SELECT CONCAT(meno,priezvisko)
FROM študent
```

alebo ekvivalentne v infixnej forme:

```
SELECT meno CONCAT priezvisko
FROM študent
```

či (obvykle) použitím lepidla:

```
SELECT meno || priezvisko
FROM študent
```

Odpoveď:

JánHraško
RuženaŠípová
AladárBaba
FerdinandMravec
JánPolienko
JurajTruľo
JanaBotková
DanaBotková
JánHlúpy
AladárMiazga
MikulášMyšiak
DonaldKáčer
JozefNámorník

Vidíme, že úspech nie je stopercentný – medzi meno a priezvisko treba ešte vložiť jednu medzeru a nový stĺpec pomenovať (aliasom):

```
SELECT CONCAT(CONCAT(meno,' '),priezvisko) AS študent
FROM študent
```

alebo prehľadnejšie:

```
SELECT meno || ' ' || priezvisko AS študent
FROM študent
```

Odpoveď:

ŠTUDENT
Ján Hraško
Ružena Šípová
Aladár Baba
Ferdinand Mravec
Ján Polienko
Juraj Truľo
Jana Botková
Dana Botková
Ján Hlúpy
Aladár Miazga
Mikuláš Myšiak
Donald Káčer
Jozef Námorník

Všimnime si, že lepiť môžeme nielen hodnoty dosadené za názvy stĺpcov, ale i konštanty (v tomto prípade medzeru), v každom prípade to však musia byť stringy.

Častým spôsobom výpisu zoznamu ľudí je použitie iba iniciálky krstného mena. Zatiaľ však nevieme získať prvé písmeno mena (zvyšok – nalepiť naň bodku, medzeru a priezvisko – nám už žiadnu starosť neurobí). Ide vlastne o špeciálny prípad získania akéhokoľvek podreťazca z reťazca. Na to je určená funkcia SUBSTR (skratka od „substring“ – „podreťazec“), ktorá má tri parametre – reťazec r, z ktorého ideme podreťazec vyberať, poradie z začiatočného znaku podreťazca v reťazci (počítané od 1) a dĺžku d podreťazca, pričom, samozrejme, $z + d - 1$ nesmie presiahnuť deklarovanú dĺžku reťazca. Napríklad:

```
VALUES SUBSTR('Mach a Šebestová',10,4)
```

je reťazec:

best

V našom prípade potrebujeme iniciálku, teda prvý znak stĺpca meno, takže výsledok bude:

```
SELECT SUBSTR(meno,1,1) || '. ' || priezvisko AS študent
FROM študent
```

Odpoveď:

ŠTUDENT
J. Hraško
R. Šípová
A. Baba
F. Mravec
J. Polienko
J. Truľo
J. Botková
D. Botková
J. Hlúpy
A. Miazga
M. Myšiak
D. Káčer
J. Námorník

Pri ďalšom častom type výpisu sa priezvisko na odlíšenie od mena píše veľkými písmenami. Práve to spôsobuje funkcia UCASE (ekvivalentnou alternatívou je UPPER) (zo slovného spojenia „upper case“ – „veľkými písmenami“), ktorá zmení všetky malé písmená na veľké, no ostatné znaky nechá tak:

```
SELECT meno || ' ' || UCASE(priezvisko) AS študent
FROM študent
```

resp.:

```
SELECT meno || ' ' || UPPER(priezvisko) AS študent
FROM študent
```

Odpoveď:

ŠTUDENT
Ján HRAŠKO
Ružena ŠÍPOVÁ
Aladár BABA
Ferdinand MRAVEC
Ján POLIENKO
Juraj TRUĽO
Jana BOTKOVÁ
Dana BOTKOVÁ
Ján HLÚPY
Aladár MIAZGA
Mikuláš MYŠIAK
Donald KÁČER
Jozef NÁMORNÍK

Komplementárnou funkciou je LCASE (alebo ekvivalentne LOWER) (z „lower case“ – „malými písmenami“), ktorá naopak všetky veľké písmená skonvertuje na malé (a ostatné znaky nezmení).

Predstavme si, že na nástenke je každé meno a priezvisko študenta napísané z vystrihnutých papierových písmen. Koľko písmen bolo treba vystrihnúť pre jednotlivých študentov? Zrejme stačí spočítať dĺžku reťazca v stĺpci meno a dĺžku reťazca v stĺpci priezvisko, jediný problém je teda určiť dĺžku reťazca. A práve na to slúži funkcia LENGTH („dĺžka“), ktorej jediným vstupom je reťazec:

```
SELECT
  meno || ' ' || priezvisko AS študent,
  LENGTH(meno) + LENGTH(priezvisko) AS počet_písmen
FROM študent
```

Odpoveď:

ŠTUDENT	POČET_PÍSMEN
Ján Hraško	9
Ružena Šípová	12
Aladár Baba	10
Ferdinand Mravec	15
Ján Polienko	11
Juraj Truľo	10
Jana Botková	11
Dana Botková	11
Ján Hlúpy	8
Aladár Miazga	12
Mikuláš Myšiak	13
Donald Káčer	11
Jozef Námorník	13

Použitím + sme načreli do veľkej skupiny **číselných** (alebo **aritmetických**) funkcií, ktorá zahŕňa všemožnú matematiku od absolútnej hodnoty až po štatistickú kovarianciu. Všetky tieto funkcie majú obvyklý význam, preto ich nebudeme ani len menovať, nieto komentovať či ilustrovať.

Teraz chcime ku každému študentovi do zátvoriek uviesť jeho študijný výsledok. No čo, žiaden problém, povieme si, veď lepiť už vieme:

```
SELECT meno || ' ' || priezvisko || ' ' || '(' || priemer || ')' AS študent
FROM študent
```

Toto však zlyhá – stĺpec priemer totiž (na rozdiel od doterajších prípadov) nemá reťazcový dátový typ. Musíme ho preto naň skonvertovať. Na konverziu slúži konštrukcia CAST(v AS d) („zmeň dátový typ (výrazu) v na (dátový typ) d“). Dobrým prostredníkom je dátový typ CHAR, na ktorý možno konvertovať každý doteraz použitý typ (i keď v prípade VARCHAR pôvodný reťazec nemôže byť pridlhý), a tiež naopak, avšak, samozrejme, len vtedy, keď sa výraz skonvertovať naozaj dá (nemôžeme napr. reťazec Osmijanko skonvertovať na celé číslo!). Možná je tiež konverzia INT na DEC (dokonca sa v prípade potreby deje automaticky), opačná je trochu zavádzajúca, lebo odsekáva desatinné miesta.

V našom príklade teda dopyt zmeníme takto (všimnime si vo výsledku trochu nepríjemnú zmenu desatinnej čiarky na bodku):

```
SELECT meno || ' ' || priezvisko || ' ' || '(' || CAST(priemer AS CHAR(4)) || ')' AS študent
FROM študent
```

Odpoveď:

ŠTUDENT
Ján Hraško (1.83)
Ružena Šípová (1.22)
Aladár Baba (2.03)
Ferdinand Mravec (1.00)
Ján Polienko (2.28)
Juraj Truľo (3.00)
Jana Botková (1.50)
Dana Botková (1.40)
Ján Hlúpy (3.00)
Aladár Miazga (2.06)
Mikuláš Myšiak (1.66)
Donald Káčer (1.83)
Jozef Námorník (2.90)

Ďalší, trochu obmedzenejší spôsob konverzie je d(v), ten však v prípade reťazca neumožňuje ovplyvniť rozsah výsledného dátového typu. Takže dopyt:

```
SELECT meno || ' ' || priezvisko || ' ' || '(' || CHAR(priemer) || ')' AS študent
FROM študent
```

s odpoveďou:

ŠTUDENT
Ján Hraško (1.83)
Ružena Šípová (1.22)
Aladár Baba (2.03)
Ferdinand Mravec (1.00)
Ján Polienko (2.28)
Juraj Truľo (3.00)
Jana Botková (1.50)
Dana Botková (1.40)
Ján Hlúpy (3.00)
Aladár Miazga (2.06)
Mikuláš Myšiak (1.66)
Donald Káčer (1.83)
Jozef Námorník (2.90)

musíme upraviť tak, aby sme sa zbavili nadbytočnej medzery medzi priemerom a zátvorkou. Je to vhodná príležitosť uviesť ďalšiu stringovú funkciu RTRIM („trim“ znamená „orezať“, „r“ je skratka pre „right“ – „sprava“), ktorá oseká všetky medzery, ktorými sa prípadne reťazec končí:

```
SELECT meno || ' ' || priezvisko || ' ' || '(' || RTRIM(CHAR(priemer)) || ')' AS študent
FROM študent
```

s odpoveďou:

ŠTUDENT
Ján Hraško (1.83)
Ružena Šípová (1.22)
Aladár Baba (2.03)
Ferdinand Mravec (1.00)
Ján Polienko (2.28)
Juraj Truľo (3.00)
Jana Botková (1.50)
Dana Botková (1.40)
Ján Hlúpy (3.00)
Aladár Miazga (2.06)
Mikuláš Myšiak (1.66)
Donald Káčer (1.83)
Jozef Námorník (2.90)

Analogicky pracuje funkcia LTRIM, ktorá odstraňuje prípadné medzery zľava. Dodajme, že výsledkom LTRIM i RTRIM po aplikovaní na prázdnu hodnotu je opäť NULL.

Ak by sme chceli vypisovať priemer len na jedno desatinné miesto, popri klasickom SUBSTR môžeme použiť konverziu v tvare DEC(c,x,y), čím vznikne zodpovedajúce číslo typu DEC(x,y). Takže:

```
SELECT meno || ' ' || priezvisko || ' ' || '(' || RTRIM(CHAR(DEC(priemer,2,1))) || ')' AS študent
FROM študent
```

Odpoveď:

ŠTUDENT
Ján Hraško (1.8)
Ružena Šípová (1.2)
Aladár Baba (2.0)
Ferdinand Mravec (1.0)
Ján Polienko (2.2)
Juraj Truľo (3.0)
Jana Botková (1.5)
Dana Botková (1.4)
Ján Hlúpy (3.0)
Aladár Miazga (2.0)
Mikuláš Myšiak (1.6)
Donald Káčer (1.8)
Jozef Námorník (2.9)

No, ktovieako dobre to nedopadlo, niektoré čísla nie sú zaokrúhlené správne (napr. u Polienka). Preto tu použijeme funkciu ROUND („zaokrúhli“), ktorá zaokrúhli číslo zo svojho prvého argumentu na počet miest v druhom argumente (pričom nula znamená zaokrúhlenie na celé číslo a záporné číslo na príslušný rád pred desatinnou čiarkou). V našom prípade teda ROUND(priemer,1):

```
SELECT meno || ' ' || priezvisko || ' ' || '(' || RTRIM(CHAR(DEC(ROUND(priemer,1),2,1))) || ')' AS študent
FROM študent
```

Odpoveď:

ŠTUDENT
Ján Hraško (1.8)
Ružena Šípová (1.2)
Aladár Baba (2.0)
Ferdinand Mravec (1.0)
Ján Polienko (2.3)
Juraj Truľo (3.0)
Jana Botková (1.5)
Dana Botková (1.4)
Ján Hlúpy (3.0)
Aladár Miazga (2.1)
Mikuláš Myšiak (1.7)
Donald Káčer (1.8)
Jozef Námorník (2.9)

Hneď je to lepšie!

Keď už sme pri konverziách, všimnime si, že i keď sme do databázy vkladali dátumy v „slovenskom“ (európskom) formáte, výpis vyzerá inak – je v ISO formáte. Náš tvar dostaneme takouto konverziou:

```
SELECT
  meno || ' ' || priezvisko AS študent,
  CHAR(dátum_narodenia,EUR) AS dátum_narodenia
FROM študent
```

Odpoveď:

ŠTUDENT	DÁTUM_NARODENIA
Ján Hraško	12.07.1987
Ružena Šípová	01.02.1984
Aladár Baba	22.01.1980
Ferdinand Mravec	03.03.1984
Ján Polienko	14.04.1982
Juraj Truľo	16.07.1979
Jana Botková	21.09.1977
Dana Botková	21.09.1977
Ján Hlúpy	NULL
Aladár Miazga	22.12.1987
Mikuláš Myšiak	06.06.1983
Donald Káčer	07.10.1982
Jozef Námorník	23.09.1981

Tu si uvedomme, že stĺpec dátum_narodenia vo výslednej tabuľke už nemá dátový typ DATE, ale CHAR. Nie je preto prekvapením, že takýto pokus usporiadať študentov podľa veku nevyjde:

```
SELECT
  meno || ' ' || priezvisko AS študent,
  CHAR(dátum_narodenia,EUR) AS dátum_narodenia
FROM študent
ORDER BY 2
```

Odpoveď:

ŠTUDENT	DÁTUM_NARODENIA
Ružena Šípová	01.02.1984
Ferdinand Mravec	03.03.1984
Mikuláš Myšiak	06.06.1983
Donald Káčer	07.10.1982
Ján Hraško	12.07.1987
Ján Polienko	14.04.1982
Juraj Truľo	16.07.1979
Jana Botková	21.09.1977
Dana Botková	21.09.1977
Aladár Baba	22.01.1980
Aladár Miazga	22.12.1987
Jozef Námorník	23.09.1981
Ján Hlúpy	NULL

Spomeňme ešte, že americký tvar (napr. 09/23/1981) dosiahneme výrazom CHAR(dátum_narodenia,USA) a ISO štandard (napr. 1981-09-23) pomocou CHAR(dátum_narodenia,ISO) (v tomto prípade vo výsledku nevidno žiaden rozdiel, zmení sa však dátový typ). Všimnime si, že usporiadanie podľa dátumov v ISO formáte spĺňa naše očakávania usporiadania dátumov. Nie nadarmo je to norma...

Skúsme urobiť to isté čo pred chvíľou s priemerom teraz aj s dátumom narodenia (použijeme však konverziu dátumu na európsky tvar, a pretože výsledok konverzie má vždy rovnakú dĺžku 10, netreba ani orezávať medzery):

```
SELECT meno || ' ' || priezvisko || ' ' || '(' || CHAR(dátum_narodenia,EUR) || ')' AS študent
FROM študent
```

Odpoveď:

ŠTUDENT
Ján Hraško (12.07.1987)
Ružena Šípová (01.02.1984)
Aladár Baba (22.01.1980)
Ferdinand Mravec (03.03.1984)
Ján Polienko (14.04.1982)
Juraj Truľo (16.07.1979)
Jana Botková (21.09.1977)
Dana Botková (21.09.1977)
NULL
Aladár Miazga (22.12.1987)
Mikuláš Myšiak (06.06.1983)
Donald Káčer (07.10.1982)
Jozef Námorník (23.09.1981)

Stala sa nemilá vec – chýbajúci druhoradý (lebo zátvorkový) dátum narodenia spôsobil stratu dát z celého riadku Jána Hlúpeho. Celé to zapríčinil postoj použitých funkcií k prázdnej hodnote: Keď je dátum_narodenia NULL, aj CHAR(dátum_narodenia,EUR) je NULL, a potom bude NULL aj celý výraz. A takto sa správajú všetky doteraz spomínané stĺpcové funkcie.

Je však škoda prísť pre nejaký málo podstatný údaj v zátvorkách o dôležité dáta, budeme preto musieť prázdnu hodnotu „odprázdniť“. Pomôžeme si na to určenou funkciou COALESCE („splynúť“, „spojiť sa“)

alebo ekvivalentne VALUE („hodnota“), ktorá môže mať ľubovoľný počet argumentov a vráti prvú neprázdnu hodnotu (ak sú prázdne všetky, nemá na výber – vráti NULL). V našom prípade teda píšme:

```
SELECT
  meno || ' ' || priezvisko || ' '
    || '(' || COALESCE(CHAR(dátum_narodenia,EUR),'neznámy dátum narodenia') || ')'
    AS študent
FROM študent
```

Odpoveď:

ŠTUDENT
Ján Hraško (12.07.1987)
Ružena Šípová (01.02.1984)
Aladár Baba (22.01.1980)
Ferdinand Mravec (03.03.1984)
Ján Polienko (14.04.1982)
Juraj Truľo (16.07.1979)
Jana Botková (21.09.1977)
Dana Botková (21.09.1977)
Ján Hlúpy (neznámy dátum narodenia)
Aladár Miazga (22.12.1987)
Mikuláš Myšiak (06.06.1983)
Donald Káčer (07.10.1982)
Jozef Námorník (23.09.1981)

Tento náš výsledok má však ešte jeden malý nedostatok. Aj keď sa občas používa zápis napr. 01.02.1984, predsa len prirodzenejšia forma je 1.2.1984, resp. spisovne s medzerami za bodkami 1. 2. 1984. Ukážeme si teda spôsob, ako ju získať. Každá hodnota typu DATE vlastne pozostáva z troch celých čísel, ktoré z neho možno extrahovať funkciami YEAR („rok“), MONTH („mesiac“) a DAY („deň“); asi netreba zdôrazňovať, že aj ony pri vstupe NULL vrátia opäť NULL. Keď tieto extrakty konvertujeme na CHAR a orežeme zbytočné medzery, po ich konkatenácii (včítane bodiek a medzier) dostaneme požadovaný dátum:

```
SELECT
  meno || ' ' || priezvisko || ' '
    || '('
    || COALESCE(
      RTRIM(CHAR(DAY(dátum_narodenia))) || '. ' ||
      RTRIM(CHAR(MONTH(dátum_narodenia))) || '. ' ||
      RTRIM(CHAR(YEAR(dátum_narodenia))),
      'neznámy dátum narodenia')
    || ')'
    AS študent
FROM študent
```

Odpoveď:

ŠTUDENT
Ján Hraško (12. 7. 1987)
Ružena Šípová (1. 2. 1984)
Aladár Baba (22. 1. 1980)
Ferdinand Mravec (3. 3. 1984)
Ján Polienko (14. 4. 1982)
Juraj Truľo (16. 7. 1979)
Jana Botková (21. 9. 1977)
Dana Botková (21. 9. 1977)
Ján Hlúpy (neznámy dátum narodenia)
Aladár Miazga (22. 12. 1987)
Mikuláš Myšiak (6. 6. 1983)
Donald Káčer (7. 10. 1982)
Jozef Námorník (23. 9. 1981)

Skúsme teraz zistiť, na ktorý deň v týždni tieto dátumy narodení pripadajú. Pomôže funkcia DAYOFWEEK („deň týždňa“), vzhľadom na americké poradie dní v týždni však 1 znamená nedeľu, 2 pondelok, atď.. Teda:

```
SELECT
  meno || ' ' || priezvisko AS študent,
  DAYOFWEEK(dátum_narodenia) AS číslo_dňa_narodenia
FROM študent
```

Odpoveď:

ŠTUDENT	ČÍSLO_DŇA_NARODENIA
Ján Hraško	1
Ružena Šípová	4
Aladár Baba	3
Ferdinand Mravec	7
Ján Polienko	4
Juraj Truľo	2
Jana Botková	4
Dana Botková	4
Ján Hlúpy	NULL
Aladár Miazga	3
Mikuláš Myšiak	2
Donald Káčer	5
Jozef Námorník	4

Podobný význam má funkcia DAYOFYEAR („deň roka“), ktorá určí absolútne poradie dňa v príslušnom roku. Napr.:

```
VALUES DAYOFYEAR('2004-03-01')
```

dáva:

61

a:

```
VALUES DAYOFYEAR('2005-03-01')
```

zas:

60

lebo 1. marec je 61. deň roku 2004, ale 60. deň roku 2005.

Predchádzajúci výpis môžeme odcloniť od (aj tak u nás nezvyklého) číslovania dní v týždni nahradením čísel príslušnými názvami – napr. namiesto `1` bude vo výpise reťazec `'nedeľa'` a namiesto `5` bude `'štvrtok'`. Poslúži na to zabudovaná funkcia DAYNAME („meno dňa“) s dátumovým argumentom, ktorá je však závislá od miestnych nastavení servera, kde je databáza umiestnená. Takže v našom prípade stačí použiť príkaz:

```
SELECT
  meno || ' ' || priezvisko AS študent,
  DAYNAME(dátum_narodenia) AS deň_narodenia
FROM študent
```

Odpoveď:

ŠTUDENT	DEŇ_NARODENIA
Ján Hraško	nedeľa
Ružena Šípová	streda
Aladár Baba	utorok
Ferdinand Mravec	sobota
Ján Polienko	streda
Juraj Truľo	pondelok
Jana Botková	streda
Dana Botková	streda
Ján Hlúpy	NULL
Aladár Miazga	utorok
Mikuláš Myšiak	pondelok
Donald Káčer	štvrtok
Jozef Námorník	streda

Veľmi podobne funguje funkcia MONTHNAME („názov mesiaca“) vracajúca názov mesiaca (opäť v nastavenom jazyku), takže príkaz:

```
SELECT
  meno || ' ' || priezvisko AS študent,
  MONTHNAME(dátum_narodenia) AS mesiac_narodenia
FROM študent
```

dáva odpoveď:

ŠTUDENT	MESIAC_NARODENIA
Ján Hraško	júl
Ružena Šípová	február
Aladár Baba	január
Ferdinand Mravec	marec
Ján Polienko	apríl
Juraj Truľo	júl
Jana Botková	september
Dana Botková	september
Ján Hlúpy	NULL
Aladár Miazga	december
Mikuláš Myšiak	jún
Donald Káčer	október
Jozef Námorník	september

Keď však chceme mať výsledné mesiace v nejakom inom jazyku, musíme si pomôcť inak. Ťažko totiž čakať, že napríklad Dobšinského Veľký Sečeň a jeho bratia môžu byť výsledkom nejakej zabudovanej funkcie, ak ich chceme mať vo výsledku dopytu, musia byť jeho súčasťou. Použijeme tu konštrukciu CASE („prípad(y)“), ktorá má podobný význam ako napr. v jazyku Pascal:

```
SELECT
  meno || ' ' || priezvisko AS študent,
  CASE MONTH(dátum_narodenia)
    WHEN 1 THEN 'Veľký Sečeň'
    WHEN 2 THEN 'Malý Sečeň'
    WHEN 3 THEN 'Brezeň'
    WHEN 4 THEN 'Dubeň'
    WHEN 5 THEN 'Traveň'
    WHEN 6 THEN 'Lipeň'
    WHEN 7 THEN 'Klaseň'
    WHEN 8 THEN 'Srpeň'
    WHEN 9 THEN 'Jaseň'
    WHEN 10 THEN 'Ruješ'
    WHEN 11 THEN 'Studeň'
    WHEN 12 THEN 'Mrazeň'
  END AS mesiac_narodenia
FROM študent
```

Odpoveď:

ŠTUDENT	MESIAC_NARODENIA
Ján Hraško	Klaseň
Ružena Šípová	Malý Sečeň
Aladár Baba	Veľký Sečeň
Ferdinand Mravec	Brezeň
Ján Polienko	Dubeň
Juraj Truľo	Klaseň
Jana Botková	Jaseň
Dana Botková	Jaseň
Ján Hlúpy	NULL
Aladár Miazga	Mrazeň
Mikuláš Myšiak	Lipeň
Donald Káčer	Rujeň
Jozef Námorník	Jaseň

Všimnime si lepšie syntax tohto konštruktu. Po kľúčovom slove `CASE` nasleduje výraz v, ktorého hodnota nás zaujíma. Potom nasledujú jednotlivé riadky vo forme `WHEN` x `THEN` y, ktoré hovoria, že „ak má v hodnotu x, tak výsledok bude y“. Konštrukcia sa končí slovom `END` („koniec“) (za ktorým ešte môže byť alias práve definovaného stĺpca). Ako nepovinný posledný riadok sa môže uviesť `ELSE` („inak“) y, ktorý hovorí, že ak výraz nenadobúda žiadnu z hodnôt v riadkoch začínajúcich sa `WHEN`, výsledok bude mať hodnotu y. Ak však `ELSE` v konštrukcii nefiguruje a výraz nenadobúda žiadnu hodnotu vymenovanú za niektorým `WHEN`, výsledná hodnota je prázdna.

Aby sme sa teda prázdnej hodnote v riadku Jána Hlúpeho vyhli, dopyt upravíme takto:

```
SELECT
  meno || ' ' || priezvisko AS študent,
  CASE MONTH(dátum_narodenia)
    WHEN 1 THEN 'Veľký Sečeň'
    WHEN 2 THEN 'Malý Sečeň'
    WHEN 3 THEN 'Brezeň'
    WHEN 4 THEN 'Dubeň'
    WHEN 5 THEN 'Traveň'
    WHEN 6 THEN 'Lipeň'
    WHEN 7 THEN 'Klaseň'
    WHEN 8 THEN 'Srpeň'
    WHEN 9 THEN 'Jaseň'
    WHEN 10 THEN 'Rujeň'
    WHEN 11 THEN 'Studeň'
    WHEN 12 THEN 'Mrazeň'
    ELSE 'neznámy mesiac narodenia'
  END AS mesiac_narodenia
FROM študent
```

Odpoveď:

ŠTUDENT	MESIAC_NARODENIA
Ján Hraško	Klaseň
Ružena Šípová	Malý Sečeň
Aladár Baba	Veľký Sečeň
Ferdinand Mravec	Brezeň
Ján Polienko	Dubeň
Juraj Truľo	Klaseň
Jana Botková	Jaseň
Dana Botková	Jaseň
Ján Hlúpy	neznámy mesiac narodenia
Aladár Miazga	Mrazeň
Mikuláš Myšiak	Lipeň
Donald Káčer	Rujeň
Jozef Námorník	Jaseň

Na záver dodajme, že `CASE` môže mať aj inú syntax, keď za `CASE` nie je žiaden výraz, ale hneď prvé `WHEN`. Za každým `WHEN` je potom namiesto hodnoty celá podmienka, pričom tieto podmienky sa nemusia navzájom vylučovať. V takom prípade berieme do úvahy prvú vyhovujúcu podmienku. Táto verzia má teda širšie možnosti.

Napríklad ak chceme zistiť, kto sa narodil cez veľké prázdniny alebo cez víkend, pomôže dopyt:

```
SELECT
  meno || ' ' || priezvisko AS študent,
  CASE
    WHEN MONTH(dátum_narodenia) = 7 THEN 'prázdniny - júl'
    WHEN MONTH(dátum_narodenia) = 8 THEN 'prázdniny - august'
    WHEN DAYOFWEEK(dátum_narodenia) = 1 THEN 'víkend - nedeľa'
    WHEN DAYOFWEEK(dátum_narodenia) = 7 THEN 'víkend - sobota'
  END AS typ_dňa_narodenia
FROM študent
```

Odpoveď:

ŠTUDENT	TYP_DŇA_NARODENIA
Ján Hraško	prázdniny - júl
Ružena Šípová	NULL
Aladár Baba	NULL
Ferdinand Mravec	víkend - sobota
Ján Polienko	NULL
Juraj Truľo	prázdniny - júl
Jana Botková	NULL
Dana Botková	NULL
Ján Hlúpy	NULL
Aladár Miazga	NULL
Mikuláš Myšiak	NULL
Donald Káčer	NULL
Jozef Námorník	NULL

Ak však vymeníme poradie víkendových a prázdninových dní, výsledok sa zmení:

```
SELECT
  meno || ' ' || priezvisko AS študent,
  CASE
    WHEN DAYOFWEEK(dátum_narodenia) = 1 THEN 'víkend - nedeľa'
    WHEN DAYOFWEEK(dátum_narodenia) = 7 THEN 'víkend - sobota'
    WHEN MONTH(dátum_narodenia) = 7 THEN 'prázdniny - júl'
    WHEN MONTH(dátum_narodenia) = 8 THEN 'prázdniny - august'
  END AS typ_dňa_narodenia
FROM študent
```

Odpoveď:

ŠTUDENT	TYP_DŇA_NARODENIA
Ján Hraško	víkend - nedeľa
Ružena Šípová	NULL
Aladár Baba	NULL
Ferdinand Mravec	víkend - sobota
Ján Polienko	NULL
Juraj Truľo	prázdniny - júl
Jana Botková	NULL
Dana Botková	NULL
Ján Hlúpy	NULL
Aladár Miazga	NULL
Mikuláš Myšiak	NULL
Donald Káčer	NULL
Jozef Námorník	NULL

Ján Hraško totiž spĺňa dve z týchto podmienok a tentoraz sa ich poradie vymenilo.

Medzi naoko zbytočné dátumové funkcie patrí aj DAYS („dni“), ktorá vráti počet dní od hypotetického 1. januára roku Pána 1. Keď však dve takéto hodnoty od seba odčítame, problematické časové intervaly sa navzájom vykrátia, a výsledok je potom počet dní medzi dvoma danými dátumami. Napríklad počet dní medzi narodením Aladára Babu (22. 1. 1980) a Ferdinanda Mravca (3. 3. 1984) zistíme dopytom:

```
VALUES DAYS(DATE('3.3.1984')) - DAYS(DATE('22.1.1980'))
```

Odpoveď:

```
1502
```

Druhý je teda od prvého mladší o 1502 dní. Ak chceme zistiť, koľko je to v rokoch, mesiacoch a dňoch, tieto dva dátumy odčítame priamo (áno, ide to!):

```
VALUES DATE('3.3.1984') - DATE('22.1.1980')
```

Odpoveď:

```
40112
```

Výsledok má umelý dátový typ DECIMAL(8), ktorého posledné dve cifry udávajú počet dní, ďalšie dve od konca počet mesiacov a začiatočné štyri (v našom prípade len jedna) zas počet rokov. Aladár Baba je teda od Ferdinanda Mravca starší o 4 roky, 1 mesiac a 12 dní. Tieto tri údaje vieme z výsledku dokonca extrahovať, a to opäť použitím (ďalších verzií) funkcií YEAR („rok“), MONTH („mesiac“) a DAY („deň“). Takže napríklad:

```
VALUES YEAR(DATE('3.3.1984') - DATE('22.1.1980'))
```

nám vráti vekový rozdiel týchto dvoch študentov v (celých) rokoch:

```
4
```

Za zmienku stojí aj ďalšia z funkcií rovnakého mena DATE („dátum“), ktorá je k funkcii DAYS inverzná. Ak teda chceme zistiť, kedy uplynie prvých sto dní nového vedenia univerzity nastúpivšieho 1. septembra 2004, stačí napísať príkaz:

```
VALUES DATE(DAYS('2004-09-01')+100)
```

Odpoveď:

```
2004-12-10
```

Takže kritizovať ho môžeme od 10. decembra tohože roku. Je to naše ľudské právo.

1.2.Ú Úlohy

1 Ako sme neraz videli, reťazce sú obvykle zarovnávané vľavo, no máme už dostatočný arzenál na to, aby sme to vedeli v prípade potreby zmeniť. Čo teda treba urobiť, keď chceme trebárs priezviská študentov zarovnať vpravo (samozrejme, za predpokladu neproporcionality písma vo výsledku)?

1.3 Výpisy záznamov spĺňajúcich nejakú podmienku

1.3.1 Základné porovnania

Doteraz sme sa zaoberali výpisom všetkých záznamov (či už priamo, alebo v nejakej upravenej forme). Často je však vhodné vypísať iba niektoré z nich – také, ktoré vyhovujú istej podmienke. Na to ale nepostačuje doterajšia konštrukcia SELECT-FROM, treba ju doplniť o podmienkovú klauzulu WHERE (v preklade „kde“).

Predstavme si, že chceme vypísať iba študentov 1. ročníka (teda tých, ktorých záznam má v stĺpci ročník hodnotu 1). Poslúži takýto dopyt:

```
SELECT *
FROM študent
WHERE ročník = 1
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ružena	Šípová	žena	1984-02-01	1	1,22
Juraj	Truľo	muž	1979-07-16	1	3,00

Tento príkaz možno čítať „vyber všetky záznamy z (tabuľky) študent, kde (t. j. v ktorých) platí podmienka ročník = 1“.

Aj takéto vybrané záznamy možno usporiadať už známym spôsobom – pomocou ORDER BY, a to, ako sme už povedali, vždy iba raz – na konci príkazu. (V ďalšom texte už preto nebudeme možnosť usporiadania špeciálne zdôrazňovať.) Napríklad ak chceme usporiadať prvákov podľa prospechu, použijeme príkaz:

```
SELECT *
FROM študent
WHERE ročník = 1
ORDER BY priemer
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ružena	Šípová	žena	1984-02-01	1	1,22
Ján	Hraško	muž	1987-07-12	1	1,83
Juraj	Truľo	muž	1979-07-16	1	3,00

Okrem rovnosti (=) možno v podmienke použiť aj ďalšie porovnania – menší (<), väčší (>), menší alebo rovný (<=), väčší alebo rovný (>=) a rôzny (<>, resp. ekvivalentne !=). Tieto porovnania možno použiť nielen pri číselných typoch, ale i pri dátumoch či reťazcoch. Napríklad študentov po treťom ročníku získame dopytom:

```
SELECT *
FROM študent
WHERE ročník > 3
```

alebo (vzhľadom na celočíselnosť stĺpca ročník) ekvivalentne:

```
SELECT *
FROM študent
WHERE ročník >= 4
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Polienko	muž	1982-04-14	5	2,28
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83

Porovnávať môžeme nielen jednotlivé hodnoty, ale aj celé usporiadané n-tice (skrátene „**tica**“) (pravdaže, ich dĺžky (n) a dátové typy ich položiek musia korešpondovať). Napríklad na vypísanie prváčok postačuje takýto príkaz:

```
SELECT *
FROM študent
WHERE (ročník, pohlavie) = (1, 'žena')
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ružena	Šípová	žena	1984-02-01	1	1,22

Dobre si všimnime odpoveď na nasledujúcu naoko bezproblémovú podmienku:

```
SELECT *
FROM študent
WHERE dátum_narodenia = dátum_narodenia
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ružena	Šípová	žena	1984-02-01	1	1,22
Aladár	Baba	muž	1980-01-22	2	2,03
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Ján	Polienko	muž	1982-04-14	5	2,28
Juraj	Truľo	muž	1979-07-16	1	3,00
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40
Aladár	Miazga	muž	1987-12-22	3	2,06
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Jozef	Námorník	muž	1981-09-23	2	2,90

Spočítajme študentov... Máme?... Jeden chýba! Hmm, čudné... Tak ešte raz... Zase dvanásť! Ale ako je to možné? Malo by ich byť všetkých trinásť, veď tá podmienka je triviálna! A môžeme vziať jed na to, že je v tom zase namočený ten nešťastný Hlúpy Jano. Ešte väčšie prekvapenie je však to, že chýba aj pri negácii tejto podmienky:

```
SELECT *
FROM študent
WHERE dátum_narodenia != dátum_narodenia
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER

Náš enfant terrible síce zmizol, ale objavil sa zaujímavý paradox, že napr. porovnania `=` a `!=` nemusia byť navzájom komplementárne. Tieto problémy spôsobuje prázdna hodnota. Jej zavedením sa totiž naša stará dobrá dvojhodnotová logika s hodnotami *pravda* a *nepravda* zmení na **trojhodnotovú** s treťou hodnotou *neznáme*, a výsledkom porovnania prázdnej hodnoty s hocičím je potom práve táto nová hodnota. Zdôraznime,

že to platí aj pre (ktorékoľvek) porovnanie dvoch prázdnych hodnôt. Ono je to vlastne celkom logické, veď tie dve prázdne hodnoty nemusia znamenať to isté – ak by sme mali ešte jedného nešťastníka, ktorý nevie svoj dátum narodenia, a teda aj on by mal v stĺpci `dátum_narodenia` `NULL`, vyplývalo by z toho azda, že sa museli narodiť v ten istý deň? (I keď by sme mohli namietať, že v našom prípade je na oboch stranách porovnania identický výraz...)

Ako teda získať riadky s/bez `NULL`? Odpoveďou je existencia špeciálnych porovnaní – otázky na prázdnosť hodnoty `IS NULL` („je prázdny“) a jej negácie `IS NOT NULL` („je neprázdny“). Ak teda chceme zistiť študentov, ktorých dátumy narodenia v databáze chýbajú, napíšeme dopyt:

```
SELECT *
FROM študent
WHERE dátum_narodenia IS NULL
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hlúpy	muž	NULL	2	3,00

A máme ho!

Teraz si už môžeme všimnúť, že zápis `COALESCE(`x_1`,`x_2`)` je vlastne skratkou pre výraz:

```
CASE
  WHEN x1 IS NOT NULL THEN x1
  ELSE x2
END
```

a výraz `COALESCE(`x_1`,`x_2`,...,`x_n`)` je definovaný rekurzívne:

```
CASE
  WHEN x1 IS NOT NULL THEN x1
  ELSE COALESCE(x2,...,xn)
END
```

A keď sme sa už takto rozohnili, pridajme ako malý bonus ešte jednu skratku pre príkaz `CASE` v špeciálnom tvare. Ide o dvojargumentovú funkciu `NULLIF` („prázdna hodnota, ak (sa rovnajú)“), pričom `NULLIF(`x_1`,`x_2`)` znamená:

```
CASE
  WHEN x1 = x2 THEN NULL
  ELSE x1
END
```

Ak teda potrebujeme do časti `SELECT` dostať prázdnu hodnotu typu `INT` (napísať priamo `NULL` je neprípustné, lebo nemá určený dátový typ), stačí uviesť napríklad `NULLIF(1,1)`.

Dodajme, že v elementárnych porovnaniach nemusia byť iba mená stĺpcov a konštanty, rovnako dobre tam zafungujú aj ľubovoľné (pravdaže, korektne zostrojené) výrazy. Jedinou podmienkou je kompatibilita dátových typov porovnávaných výrazov. Napríklad študentov, ktorí majú obe iniciálky rovnaké, dostaneme takto:

```
SELECT *
FROM študent
WHERE SUBSTR(meno,1,1) = SUBSTR(priezvisko,1,1)
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Mikuláš	Myšiak	muž	1983-06-06	5	1,66

A na záver ešte jeden postreh o benevolentnosti porovnávania reťazcov ku koncovým medzerám. Ťažko tomu uveriť, ale podmienky v nasledujúcich dvoch príkazoch sú naozaj ekvivalentné:

```
SELECT *
FROM študent
WHERE meno = 'Ján'
```

```
SELECT *
FROM študent
WHERE meno = 'Ján '
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ján	Polienko	muž	1982-04-14	5	2,28
Ján	Hlúpy	muž	NULL	2	3,00

1.3.2 Zložitejšie porovnania a žolíky

Našu pozornosť si zaslúžia aj ďalšie tri druhy porovnania:

Ak má byť nejaká hodnota z istého (uzavretého) intervalu, použijeme konštrukt BETWEEN-AND („medzi (niečím) a (niečím)“). Napríklad ak máme záujem o študentov narodených medzi 22. 1. 1980 a 1. 2. 1984, použijeme príkaz:

```
SELECT *
FROM študent
WHERE dátum_narodenia BETWEEN '22.1.1980' AND '1.2.1984'
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ružena	Šípová	žena	1984-02-01	1	1,22
Aladár	Baba	muž	1980-01-22	2	2,03
Ján	Polienko	muž	1982-04-14	5	2,28
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Jozef	Námorník	muž	1981-09-23	2	2,90

Všimnime si, že interval v BETWEEN zahŕňa aj krajné hodnoty – v odpovedi sú aj Ružena Šípová a Aladár Baba, ktorých dátumy narodenia sa zhodujú s hranicami.

Ak má mať stĺpec hodnotu z istého zoznamu, poslúži konštrukt IN („v“), za ktorým nasleduje zoznam hodnôt v zátvorkách. Napríklad ak potrebujeme iba druhákov a piatakov, použijeme dopyt:

```
SELECT *
FROM študent
WHERE ročník IN (2,5)
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Aladár	Baba	muž	1980-01-22	2	2,03
Ján	Polienko	muž	1982-04-14	5	2,28
Ján	Hlúpy	muž	NULL	2	3,00
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Jozef	Námorník	muž	1981-09-23	2	2,90

Tretím špeciálnym porovnaním je LIKE („podobné“), pri ktorom ide o prácu s **maskami** (t. j. len čiastočne definovanými reťazcami) využitím tzv. **žolíkov**, často nazývaných aj pôvodným názvom „**wildcards**“ – špeciálnych

reťazcových premenných % a _. Prvá z nich nahradzuje ľubovoľný reťazec (včítane prázdneho), druhá ľubovoľný (ale práve jeden) znak. (Je to len iné vyjadrenie klasických žolíkov *, resp. ?.)

Ak chceme vypísať len študentov, ktorých meno sa začína na písmeno J, dopyt sformulujeme takto:

```
SELECT *
FROM student
WHERE meno LIKE 'J%'
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ján	Polienko	muž	1982-04-14	5	2,28
Juraj	Truľo	muž	1979-07-16	1	3,00
Jana	Botková	žena	1977-09-21	4	1,50
Ján	Hlúpy	muž	NULL	2	3,00
Jozef	Námorník	muž	1981-09-23	2	2,90

Študentov s priezviskom obsahujúcim (kdekoľvek) skupinu písmen ko dostaneme takto:

```
SELECT *
FROM student
WHERE priezvisko LIKE '%ko%'
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ján	Polienko	muž	1982-04-14	5	2,28
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40

Študenti, ktorých predposledné písmeno mena je á, budú vypísaní po zadaní dopytu:

```
SELECT *
FROM student
WHERE meno LIKE '%á_'
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Aladár	Baba	muž	1980-01-22	2	2,03
Ján	Polienko	muž	1982-04-14	5	2,28
Ján	Hlúpy	muž	NULL	2	3,00
Aladár	Miazga	muž	1987-12-22	3	2,06
Mikuláš	Myšiak	muž	1983-06-06	5	1,66

Vidíme teda, že oba žolíky môžeme aj kombinovať. Pre zaujímavosť dodajme, že masky '_%' aj '%_' vyjadrujú tú istú množinu – všetky neprázdne reťazce.

Pamätá sa aj na situáciu, keď treba vyhľadať všetky výrazy obsahujúce znak % (analogicky je to so znakom _). Nemôžeme totiž napísať LIKE '%%%' – databázový stroj by zrejme nepochopil (a právom), ktorý znak % znamená naozaj percento a ktorý je žolíkom. Riešením je zvoliť nejaký tzv. ESCAPE („únik“) – **únikový znak**, napr. \, a potom napísať LIKE '%\%%' ESCAPE '\'. Tak bude zrejmé, že ten znak %, ktorému predchádza únikový znak, je naozaj percento a ostatné sú žolíky.

Porovnanie LIKE môžeme použiť aj bez žolíkov. Ak si spomenieme na neobvyklé správanie = pri reťazcoch, tak tu je situácia diametrálne odlišná. Kým príkaz:

```
SELECT *
FROM študent
WHERE meno LIKE 'Ján'
```

vráti odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ján	Polienko	muž	1982-04-14	5	2,28
Ján	Hlúpy	muž	NULL	2	3,00

pri príkaze:

```
SELECT *
FROM študent
WHERE meno LIKE 'Ján '
```

je to očakávaná prázdna tabuľka:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER

1.3.3 Logické spojky

Všetky tieto elementárne podmienky možno skladať do zložitejších pomocou logických spojok – unárneho NOT (negácia – „nie“) a binárnych AND (konjunkcia – „a“) a OR (disjunkcia – „alebo“) a zátvoriek štandardným spôsobom, pričom NOT má (pri zátvorkovaní) najvyššiu a OR najnižšiu prioritu.

Napríklad ak chceme tretiakov s priemerom medzi 2 a 2,5, použijeme dopyt:

```
SELECT *
FROM študent
WHERE
      ročník = 3
  AND priemer BETWEEN 2 AND 2.5
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Aladár	Miazga	muž	1987-12-22	3	2,06

Všetkých ostatných tretiakov dostaneme takto:

```
SELECT *
FROM študent
WHERE
      ročník = 3
  AND NOT priemer BETWEEN 2 AND 2.5
```

alebo ekvivalentne:

```
SELECT *
FROM študent
WHERE
      ročník = 3
  AND (priemer < 2 OR priemer > 2.5)
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ferdinand	Mravec	muž	1984-03-03	3	1,00

Všimnime si uzátvorkovanie v druhej verzii. Ak by zátvorky chýbali, vzhľadom na prioritu spojok by sa elementárna podmienka `ročník = 3` vzťahovala len na podmienku `priemer < 2`, a do odpovede by sa tak dostali aj študenti ostatných ročníkov, ktorých priemer je viac než 2,5:

```
SELECT *
FROM študent
WHERE
      ročník = 3
  AND priemer < 2 OR priemer > 2.5
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Juraj	Truľo	muž	1979-07-16	1	3,00
Ján	Hlúpy	muž	NULL	2	3,00
Jozef	Námorník	muž	1981-09-23	2	2,90

Pripomeňme, že aj tu pracujeme s trojhodnotovou logikou. Do výsledku dopytu sa potom dostanú všetky tie záznamy, ktorých pravdivostná hodnota podmienky vo `WHERE` je *pravda*, teda nie *nepravda*, ale ani *neznáme*. Naznačme tabuľkami, ako sa v nej správajú spojky `NOT`, `AND` a `OR`:

	NOT
pravda	*nepravda*
nepravda	*pravda*
neznáme	*neznáme*

AND	*pravda*	*nepravda*	*neznáme*
pravda	*pravda*	*nepravda*	*neznáme*
nepravda	*nepravda*	*nepravda*	*nepravda*
neznáme	*neznáme*	*nepravda*	*neznáme*

OR	*pravda*	*nepravda*	*neznáme*
pravda	*pravda*	*pravda*	*pravda*
nepravda	*pravda*	*nepravda*	*neznáme*
neznáme	*pravda*	*neznáme*	*neznáme*

1.4 Agregácie

1.4.1 Jednoduché agregácie

Niekedy (a to dosť často) nás zaujímajú nielen vlastnosti jednotlivých záznamov, ale i vlastnosti nejakej ich množiny, tzv. **agregácie**, napr. všetkých riadkov ako celku. Na ich výpočet slúžia **agregačné funkcie**, typickým príkladom je funkcia COUNT („počet“), ktorá zistí počet záznamov v tabuľke. Ak nás teda zaujíma, koľko študentov máme, napíšeme:

```
SELECT COUNT(*) AS počet_študentov
FROM študent
```

Odpoveď:

POČET_ŠTUDENTOV
13

Znak * opäť znamená všetky stĺpce. Ak by bola tabuľka prázdna, výsledkom funkcie COUNT by bola nula.

Namiesto * možno napísať názov hocijakého stĺpca, prípadne aj zložitejší výraz. Výsledkom je potom počet takých záznamov, pre ktoré je hodnota tohto výrazu neprázdna. Keďže každý študent má meno i priezvisko, namiesto predchádzajúceho výrazu môžeme ekvivalentne (i keď málo zmysluplne) písať:

```
SELECT COUNT(meno) AS počet_mien
FROM študent
```

alebo hoci aj:

```
SELECT COUNT(meno || ' ' || priezvisko) AS počet_mien_s_priezviskami
FROM študent
```

Odpovede:

POČET_MIEN
13

a:

POČET_MIEN_S_PRIEZVISKAMI
13

Naproti tomu, keďže nie je známy dátum narodenia Jána Hlúpeho, má nasledujúci dopyt:

```
SELECT COUNT(dátum_narodenia) AS počet_dátumov_narodenia
FROM študent
```

inú odpoveď:

POČET_DÁTUMOV_NARODENIA
12

Mohlo by nás zaujímať, koľko rôznych krstných mien študenti majú, najprv si však zistíme, ktoré to sú. Všetky vypísať už dávno vieme:

```
SELECT meno
FROM študent
```

Odpoveď:

MENO
Ján
Ružena
Aladár
Ferdinand
Ján
Juraj
Jana
Dana
Ján
Aladár
Mikuláš
Donald
Jozef

Aby sme zabránili ich opakovaniu, hneď za SELECT vložíme slovo DISTINCT („rôzny“):

```
SELECT DISTINCT meno
FROM študent
```

Odpoveď:

MENO
Aladár
Dana
Donald
Ferdinand
Jana
Ján
Jozef
Juraj
Mikuláš
Ružena

Ak chceme zistiť ich počet, výraz DISTINCT meno vezmeme za argument funkcie COUNT:

```
SELECT COUNT(DISTINCT meno) AS počet_rôznych_mien
FROM študent
```

Odpoveď:

POČET_RÔZNYCH_MIEN
10

Dodajme, že ak je stĺpcov či výrazov viac, DISTINCT možno ho použiť tiež, no musí nasledovať bezprostredne za slovom SELECT. Hľadá totiž rôzne riadky ako celky, nie jednotlivé ich časti. Napríklad dopyt:

```
SELECT DISTINCT
  priezvisko,
  meno,
  YEAR(dátum_narodenia) AS rok_narodenia
FROM študent
```

s odpoveďou:

PRIEZVISKO	MENO	ROK_NARODENIA
Botková	Dana	1977
Botková	Jana	1977
Truľo	Juraj	1979
Baba	Aladár	1980
Námorník	Jozef	1981
Káčer	Donald	1982
Polienko	Ján	1982
Myšiak	Mikuláš	1983
Mravec	Ferdinand	1984
Šípová	Ružena	1984
Miazga	Aladár	1987
Hraško	Ján	1987
Hlúpy	Ján	NULL

neeliminuje žiaden zo záznamov, lebo všetky (ako usporiadané trojice) sú rôzne (hoci niektoré priezviská, mená i roky sa opakujú).

Ďalšími agregačnými funkciami sú MAX (skratka pre „maximum“) a MIN (skratka pre „minimum“), ktoré nájdu najväčšiu, resp. najmenšiu hodnotu svojho jediného argumentu. Ak chceme napr. zistiť dátum najmladšieho (t. j. najneskôr narodeného) študenta, píšme:

```
SELECT MAX(dátum_narodenia) AS najväčší_dátum_narodenia
FROM študent
```

Odpoveď:

NAJVÄČŠÍ_DÁTUM_NARODENIA
1987-12-22

Všimnime si, že z hľadania maxima (ale i minima) sú prázdne hodnoty vylúčené. Ak sú všetky hodnoty prázdne, tak aj maximum a minimum budú NULL.

Posledné dve agregačné funkcie sú AVG (skratka pre „average“ – „priemer“) a SUM („súčet“), ktoré rátajú (aritmetický) priemer, resp. súčet výrazu vo svojom (jedinom) argumente. Z toho vyplýva, že výraz, na ktorý aplikujeme jednu z týchto funkcií, musí byť číselného typu. (I keď jestvujú ustanovizne, kde vedia počítať priemer aj z písmen...) Priemerný prospech našich študentov teda získame dopytom:

```
SELECT AVG(priemer) AS priemerný_prospech
FROM študent
```

Odpoveď:

PRIEMERNÝ_PROSPECH
1,97769230769230769230769230769 2

Aj tu dodajme, že prázdne hodnoty sa do výpočtu SUM ani AVG vôbec nezapočítavajú – akoby v tabuľke ani neboli. Ak ich aplikujeme na prázdnu tabuľku (alebo na stĺpec len s prázdnymi hodnotami), obe dajú výsledok NULL.

Trochu nepríjemnou a mätúcou vlastnosťou funkcie AVG aplikovanej na celočíselný dátový typ je, že aj výsledok je potom celé číslo. Ak chceme naozajstný aritmetický priemer, musíme príslušný stĺpec konvertovať na desatinné číslo. Napríklad pri výpočte priemerného ročníka našich študentov nepoužijeme dopyt:

```
SELECT AVG(ročník) AS zle_počítaný_priemerný_ročník
FROM študent
```

s odpoveďou:

ZLE_POČÍTANÝ_PRIEMERNÝ_ROČNÍK
2

ale:

```
SELECT AVG(CAST (ročník AS DEC(3,2))) AS priemerný_ročník
FROM študent
```

Odpoveď:

PRIEMERNÝ_ROČNÍK
2,92307692307692307692307692307692

Vidíme teda, že pôvodná odpoveď bola naozaj značne skresľujúca.

Uvedené agregačné funkcie nemusia byť aplikované na množinu všetkých záznamov, ako to bolo doteraz, pokojne môžu byť kombinované s klauzulou `WHERE`. Napríklad ak nás zaujíma len priemerný študijný výsledok tretiakov, do množiny, na ktorú nasadíme funkciu `AVG`, dáme len študentov 3. ročníka:

```
SELECT AVG(priemer) AS priemerný_prospech_tretiakov
FROM študent
WHERE ročník = 3
```

Odpoveď:

PRIEMERNÝ_PROSPECH_TRETIAKOV
1,53000000000000000000000000000

1.4.2 Agregácie po skupinách

Doteraz sa všetky naše príklady zaoberali vlastnosťami jednej množiny (či už všetkých, alebo iba vybraných) záznamov. Môže nás však zaujímať i porovnanie vlastností viacerých paralelných **skupín** – napríklad študijný priemer jednotlivých ročníkov. Všetky záznamy (v tomto prípade študentov) podľa hodnôt príslušného tzv. **skupinovacieho výrazu** (tu stĺpca `ročník`) zoskupíme do niekoľkých skupín a potom pre každú z nich zistíme hodnotu agregačnej funkcie (tu priemer zo študijných výsledkov jednotlivých členov) tak, ako keby bola tabuľka tvorená len touto skupinou. Každá hodnota skupinovacieho výrazu teda prispeje do výslednej tabuľky jedným riadkom. Stĺpce budú dva – v prvom budú rôzne hodnoty výrazu charakterizujúce jednotlivé skupiny, v druhom hodnoty príslušných agregácií. Všimnime si, že v poslednom riadku dopytu musí byť klauzula `GROUP BY` („zoskupiť podľa“), za ktorou je spravidla zopakovaný skupinovací výraz z klauzuly `SELECT`.

V našom prípade teda dopyt vyzerá takto:

```
SELECT
  ročník,
  AVG(priemer) AS priemerný_prospech
FROM študent
GROUP BY ročník
```

Odpoveď:

ROČNÍK	PRIEMERNÝ_PROSPECH
1	2,01666666666666666666666666666
2	2,64333333333333333333333333333
3	1,53000000000000000000000000000
4	1,45000000000000000000000000000
5	1,92333333333333333333333333333

Skupinovacích výrazov môže byť aj viac – skupiny sú potom dané existujúcimi kombináciami hodnôt všetkých neagregovaných výrazov z klauzuly `SELECT` (a tie sa následne všetky zopakujú za klauzulou `GROUP BY`). Napríklad ak nás zaujímajú počty študentov podľa pohlavia aj podľa rokov narodenia (t. j. napr. odpoveď na otázku, koľko študentiek sa narodilo v roku 1977), napíšeme dopyt:

```
SELECT
  YEAR(dátum_narodenia) AS rok_narodenia,
  pohlavie,
  COUNT(*) AS počet
FROM študent
GROUP BY
  YEAR(dátum_narodenia),
  pohlavie
```

Odpoveď:

ROK_NARODENIA	POHLAVIE	POČET
1977	žena	2
1979	muž	1
1980	muž	1
1981	muž	1
1982	muž	2
1983	muž	1
1984	muž	1
1984	žena	1
1987	muž	2
NULL	muž	1

Za zmienku stojí aj modifikovaná forma tohto dopytu:

```
SELECT
  YEAR(dátum_narodenia) AS rok_narodenia,
  pohlavie,
  COUNT(*) AS počet
FROM študent
GROUP BY
  CUBE(YEAR(dátum_narodenia)),
  pohlavie
```

s odpoveďou:

ROK_NARODENIA	POHLAVIE	POČET
NULL	muž	10
NULL	žena	3
1977	žena	2
1979	muž	1
1980	muž	1
1981	muž	1
1982	muž	2
1983	muž	1
1984	muž	1
1984	žena	1
1987	muž	2
NULL	muž	1

Slovkom `CUBE` („kocka“, ale v zmysle „(hyper)kváder“) v klauzule `GROUP BY` naznačujeme, že vybrané skupinovacie výrazy chápu ako súradnice akéhosi hyperkvádra, v rámci ktorého nás zaujímajú nielen agregácie po skupinách, ale aj také, pri ktorých sú niektoré z týchto súradníc ignorované. Ide teda o tzv. **marginálne agregácie**. V odpovedi ich rozoznáme podľa toho, že v príslušných stĺpcoch je `NULL`. V našom prípade sme vybrali iba `YEAR(dátum_narodenia)`, takže navyše dostaneme marginálne počty (teda bez ohľadu na rok narodenia) mužov (hovorí o tom záznam v prvom napísanom riadku) a žien (záznam z druhého napísaného riadka). Táto ukážka je však zároveň antiukážkou: Údaj o celkovom počte mužov je totiž skreslený tým, že v poslednom vypísanom riadku máme ďalší záznam s hodnotami `NULL` a `muž`. Pri používaní skupinovania pomocou `CUBE` (a hlavne pri interpretácii výsledku) preto **musíme dávať pozor, či príslušné skupinovacie stĺpce náhodou neobsahujú prázdne hodnoty**.

Na tomto mieste zdôraznime, že (na rozdiel od `ORDER BY`) sa v klauzule `GROUP BY` nemôžu namiesto výrazov písať ich aliasy alebo ich poradové čísla. Zlyhá teda ako dopyt:

```
SELECT
  YEAR(dátum_narodenia) AS rok_narodenia,
  pohlavie,
  COUNT(*) AS počet
FROM študent
GROUP BY 1, 2
```

tak aj:

```
SELECT
  YEAR(dátum_narodenia) AS rok_narodenia,
  pohlavie,
  COUNT(*) AS počet
FROM študent
GROUP BY
  rok_narodenia,
  pohlavie
```

Pre jednotlivé skupiny môžeme naraz zistiť aj viacero agregácií. Takto napríklad zistíme jedným dopytom pre jednotlivé ročníky popri priemernom prospechu aj počet ich študentov (všimnime si, že v `GROUP BY` sa ocitol len prvý, jediný neagregovaný výraz):

```
SELECT
  ročník,
  COUNT(*) AS počet,
  AVG(priemer) AS priemerný_prospech
FROM študent
GROUP BY ročník
```

Odpoveď:

ROČNÍK	POČET	PRIEMERNÝ_PROSPECH
1	3	2,01666666666666666666666666666666
2	3	2,64333333333333333333333333333333
3	2	1,53000000000000000000000000000000
4	2	1,45000000000000000000000000000000
5	3	1,92333333333333333333333333333333

Chcime teraz nájsť mesiace, v ktorých sa narodili aspoň dvaja ľudia. Vieme už urobiť prehľad početnosti pre každý mesiac, no vidíme, že vo výpise sú aj mesiace s jedným študentom:

```
SELECT
  MONTH(dátum_narodenia) AS mesiac_narodenín,
  COUNT(*) AS počet
FROM študent
GROUP BY MONTH(dátum_narodenia)
```

Odpoveď:

MESIAC_NARODENÍN	POČET
1	1
2	1
3	1
4	1
6	1
7	2
9	3
10	1
12	1
NULL	1

Ako sa týchto nadbytočných riadkov zbaviť? Uvedomme si, že ich nemôžeme eliminovať použitím klauzuly `WHERE`. Tá totiž hovorí o jednotlivých záznamoch, no vlastnosť, ktorú skúmame, nie je vlastnosťou žiadneho

jednotlivca, ale celej skupiny. Potrebujeme analógiu WHERE, ktorá by z tohto medzivýsledku odstránila všetky riadky nevyhovujúce istej podmienke. Nachádzame ju v klauzule HAVING („majúci“), nasledovanej príslušnou podmienkou. Takže v našom prípade:

```
SELECT
  MONTH(dátum_narodenia) AS mesiac_narodenín,
  COUNT(*) AS počet
FROM študent
GROUP BY MONTH(dátum_narodenia)
HAVING COUNT(*) > 1
```

Odpoveď:

MESIAC_NARODENÍN	POČET
7	2
9	3

Žiaľ, (podobne ako pri GROUP BY) ani pri HAVING nemôžeme použiť prípadné aliasy agregovaných výrazov, keďže tie sa zrejme udeľujú až hotovému výsledku. Takže takýto dopyt by zlyhal:

```
SELECT
  MONTH(dátum_narodenia) AS mesiac_narodenín,
  COUNT(*) AS počet
FROM študent
GROUP BY MONTH(dátum_narodenia)
HAVING počet > 1
```

Dodajme, že WHERE a HAVING môžu byť spolu v jednom dopyte. Napríklad predchádzajúci prehľad, ale len pre štvrtákov dosiahneme dopytom:

```
SELECT
  MONTH(dátum_narodenia) AS mesiac_narodenín,
  COUNT(*) AS počet
FROM študent
WHERE ročník = 4
GROUP BY MONTH(dátum_narodenia)
HAVING COUNT(*) > 1
```

Odpoveď:

MESIAC_NARODENÍN	POČET
9	2

Uvedomme si, že podmienky vo WHERE a v HAVING (obvykle) nemožno miešať – `ročník = 4` je len vecou jednotlivcov (skupiny sú tentoraz podľa mesiacov) a `COUNT(*) > 1` zasa výhradne záležitosťou skupín.

1.5 Vnorené dopyty

1.5.1 Vnorené dopyty bez parametra

Ak napríklad chceme získať druhý najmenší dátum narodenia (nemusí to byť nutne dátum narodenia druhého najstaršieho človeka, lebo najstarší môžu byť aj dvaja), treba si uvedomiť, že je to minimum tých dátumov, ktoré sú väčšie ako minimum všetkých dátumov. Absolútne minimum už dostať vieme:

```
SELECT MIN(dátum_narodenia) AS najmenší_dátum_narodenia
FROM študent
```

Odpoveď:

NAJMENŠÍ_DÁTUM_NARODENIA
1977-09-21

Druhý najmenší dátum teda bude:

```
SELECT MIN(dátum_narodenia) AS druhý_najmenší_dátum_narodenia
FROM študent
WHERE dátum_narodenia > '21.9.1977'
```

Odpoveď:

DRUHÝ_NAJMENŠÍ_DÁTUM_NARODENIA
1979-07-16

Použitie konštanty však robí dopyt nevšeobecným – stačí, aby prišiel nový študent, ktorý bude ešte starší ako doteraz najstaršie Danka a Janka, a dopyt prestane byť správny – bude treba zmeniť konštantu. Jednoduchšie však bude úplne sa tejto konštanty zbaviť – nahradíme ju tzv. **vnoreným dopytom**, ktorý ju vyprodukuje:

```
SELECT MIN(dátum_narodenia) AS druhý_najmenší_dátum_narodenia
FROM študent
WHERE dátum_narodenia >
      (
        SELECT MIN(dátum_narodenia) AS najmenší_dátum_narodenia
        FROM študent
      )
```

Odpoveď:

DRUHÝ_NAJMENŠÍ_DÁTUM_NARODENIA
1979-07-16

Všimnime si, že vnorený dopyt v klauzule `WHERE` musí byť v zátvorkách a jeho výsledkom musí byť tabuľka s takými stĺpcami, aby ich počet a dátové typy korešpondovali s počtom a dátovými typmi porovnávanych hodnôt (v našom prípade je na každej strane len jedna hodnota dátového typu `DATE`). Navyše, ak sa vyskytuje v binárnom porovnávaní (ako v našom prípade), výsledná tabuľka musí mať jediný riadok. Jedine v prípade porovnania `IN` môže byť riadkov odpovede na vnorený dopyt viac.

Konštrukciu s vnoreným dopytom môžeme stupňovať. Napríklad tretí najmenší dátum narodenia môžeme nájsť takto:

```
SELECT MIN(dátum_narodenia) AS tretí_najmenší_dátum_narodenia
FROM študent
WHERE dátum_narodenia >
      (
        SELECT MIN(dátum_narodenia) AS druhý_najmenší_dátum_narodenia
        FROM študent
        WHERE dátum_narodenia >
              (
                SELECT MIN(dátum_narodenia) AS najmenší_dátum_narodenia
                FROM študent
              )
      )
```

Odpoveď:

TRETÍ_NAJMENŠÍ_DÁTUM_NARODENIA
1980-01-22

Vnorený dopyt môžeme s úspechom využiť aj pri nájdení mena a priezviska najstaršieho študenta (zatiaľ vieme získať len dátum jeho narodenia). Najprv sa v už známom vnorenom dopyte nájde dátum jeho narodenia, ten sa zaň (ako pred chvíľou) „dosadí“, a tým „vznikne“ opäť obyčajný dopyt, ktorý vráti požadované meno a priezvisko:

```
SELECT meno || ' ' || priezvisko AS najstarší_študent
FROM študent
WHERE dátum_narodenia =
      (
        SELECT MIN(dátum_narodenia) AS najmenší_dátum_narodenia
        FROM študent
      )
```

Odpoveď:

NAJSTARŠÍ_ŠTUDENT
Jana Botková
Dana Botková

Všimnime si, že sme takto našli všetkých najstarších študentov.

1.5.2 Vnorené dopyty s parametrom

Pozrime sa na ďalšiu zaujímavú aplikáciu vnorených dopytov. Už dávno vieme záznamy usporiadať podľa hodnôt nejakého stĺpca či výrazu, nevieme však zatiaľ získať ich poradové čísla. Vieme napríklad, že najlepší prospech má Ferdo a najhorší Hlúpy Jano, a chceli by sme to v odpovedi zdôrazniť spôsobom `13. Ján Hlúpy` či `1. Ferdinand Mravec`. Uvedomme si, že poradie záznamu v usporiadaní je vlastne o 1 zväčšený počet záznamov, ktoré sú v tomto usporiadaní pred ním. Poradové číslo Šípovej Ruženky s prospechom 1,22 teda zistíme dopytom:

```
SELECT COUNT(*) + 1 AS poradie_Šípovej
FROM študent
WHERE priemer < 1.22
```

Odpoveď:

PORADIE_ŠÍPOVEJ
2

Podobne s Jankom Hraškom, ktorého priemer je 1,83:

```
SELECT COUNT(*) + 1 AS poradie_Hraška
FROM študent
WHERE priemer < 1.83
```

Odpoveď:

PORADIE_HRAŠKA
6

Toto, pravdaže, môžeme urobiť s ľubovoľným študentom, vždy však iba s jedným – jeho priemer je konštanta. Vnútorný dopyt, ktorý je zovšeobecnením predchádzajúcich dvoch, potom vyzerá takto:

```
SELECT COUNT(*) + 1 AS poradie_š
FROM študent
WHERE priemer < š.priemer
```

Tento dopyt však sám osebe neprejde, lebo za výraz `š.priemer` zatiaľ nemáme čo dosadiť. Možno ho však chápať ako akúsi **procedúru**, `š.priemer` bude jej **parameter**. Napĺňať ho budeme obyčajným prechodom (ďalšieho exemplára) tabuľky `študent`. Aby sme však tieto dva exemplár tej istej tabuľky odlíšili a učinili zadosť trochu predčasnému označeniu parametra, ten vonkajší premenujeme **aliasom** práve na `š`. Alias tabuľky nasleduje bezprostredne po jej mene, prípadne (podobne ako v prípade stĺpcov) je od neho oddelený opäť slovíčkom `AS`. Na rozdiel od stĺpca **pri tabuľke s aliasom nemôžeme použiť jej pôvodné meno**. Výsledný dopyt je teda:

```
SELECT
  š.meno,
  š.priezvisko,
  š.priemer,
  (
    SELECT COUNT(*) + 1
    FROM študent
    WHERE priemer < š.priemer
  ) AS poradie
FROM študent AS š
```

Odpoveď:

MENO	PRIEZVISKO	PRIEMER	PORADIE
Ján	Hraško	1,83	6
Ružena	Šípová	1,22	2
Aladár	Baba	2,03	8
Ferdinand	Mravec	1,00	1
Ján	Polienko	2,28	10
Juraj	Truľo	3,00	12
Jana	Botková	1,50	4
Dana	Botková	1,40	3
Ján	Hlúpy	3,00	12
Aladár	Miazga	2,06	9
Mikuláš	Myšiak	1,66	5
Donald	Káčer	1,83	6
Jozef	Námorník	2,90	11

Všimnime si ešte, že na poslednom mieste sú dvaja študenti – majú totiž rovnaký priemer. To isté platí aj pre Hraška a Káčera na 6. mieste.

Aby sme splnili pôvodné zadanie, treba ešte tento výpis preformátovať. Keďže vnorený dopyt možno použiť aj v klauzule `FROM`, predchádzajúci výsledok nám dobre poslúži ako pomocná tabuľka. Musí však mať nejaký alias (tu je to `pomocná`). Nesmieme tiež zabudnúť skonvertovať jej stĺpec `poradie`:

```
SELECT RTRIM(CHAR(poradie)) || '. ' || meno || ' ' || priezvisko AS študent
FROM
(
  SELECT
    š.meno,
    š.priezvisko,
    (
      SELECT COUNT(*) + 1
      FROM študent
      WHERE priemer < š.priemer
    ) AS poradie
  FROM študent AS š
) AS pomocná
ORDER BY poradie
```

Odpoveď:

ŠTUDENT
1. Ferdinand Mravec
2. Ružena Šípová
3. Dana Botková
4. Jana Botková
5. Mikuláš Myšiak
6. Ján Hraško
6. Donald Káčer
8. Aladár Baba
9. Aladár Miazga
10. Ján Polienko
11. Jozef Námorník
12. Juraj Truľo
12. Ján Hlúpy

Pokiaľ ide o **aliasy (či už stĺpcov alebo tabuliek)**, treba ešte dodať, že **platia len počas dopytu, po jeho vykonaní zanikajú**, a prípadne premenovaná tabuľka či stĺpec má tak opäť svoje staré dobré meno.

Vráťme sa ešte k faktu, že dvojice Hraško & Káčer a Truľo & Hlúpy majú rovnaké poradové číslo. Často sa takéto umiestnenie píše nie ako jedno číslo, ale v tvare rozsahu „od – do“. Takýto tvar však, samozrejme, potrebujeme len pri tých záznamoch, ktoré sa so svojím poradím delia s iným záznamom (v našich prípadoch to budú poradia 6. – 7. a 12. – 13.), inak ostaneme pri pôvodnom tvare. Povedali sme si už, že prvé číslo označuje o jeden zväčšený počet záznamov v príslušnom zmysle „lepších“ než náš záznam. Druhé číslo potom znamená počet aspoň takých dobrých záznamov. Napríklad počet aspoň takých dobrých študentov ako Ružena Šípová s priemerom 1,22 dostaneme pomocou dopytu:

```
SELECT COUNT(*) AS poradie_Šípovej
FROM študent
WHERE priemer <= 1.22
```

Odpoveď:

PORADIE_ŠÍPOVEJ
2

A to isté s Hraškom:

```
SELECT COUNT(*) AS poradie_Hraška
FROM študent
WHERE priemer <= 1.83
```

Odpoveď:

PORADIE_HRAŠKA
7

Všimnime si, že toto poradie sa u Ruženky zhoduje s predchádzajúcim, pretože nikto iný takýto priemer nemá. U Hraška je však situácia iná, lebo rovnaký priemer má ešte niekto iný. Ak tento dopyt vnoríme do dopytu prechádzajúceho cez všetkých študentov, pričom konštantu vymeníme (podobne ako v predošlých ukážkach) za priemer aktuálneho študenta, dostávame:

```
SELECT
  š.meno,
  š.priezvisko,
  š.priemer,
  (
    SELECT COUNT(*) + 1
    FROM študent
    WHERE priemer < š.priemer
  ) AS poradie_od,
  (
    SELECT COUNT(*)
    FROM študent
    WHERE priemer <= š.priemer
  ) AS poradie_do
FROM študent AS š
```

Odpoveď:

MENO	PRIEZVISKO	PRIEMER	PORADIE_OD	PORADIE_DO
Ján	Hraško	1,83	6	7
Ružena	Šípová	1,22	2	2
Aladár	Baba	2,03	8	8
Ferdinand	Mravec	1,00	1	1
Ján	Polienko	2,28	10	10
Juraj	Truľo	3,00	12	13
Jana	Botková	1,50	4	4
Dana	Botková	1,40	3	3
Ján	Hlúpy	3,00	12	13
Aladár	Miazga	2,06	9	9
Mikuláš	Myšiak	1,66	5	5
Donald	Káčer	1,83	6	7
Jozef	Námorník	2,90	11	11

Teraz už máme všetky potrebné údaje, treba ich iba naformátovať. Ak sa hodnoty v stĺpcoch poradie_od a poradie_do zhodujú, stačí vypísať jednu (ľubovoľnú) z nich. Ak sú však rôzne, treba vypísať obe. Na toto rozlíšenie použijeme CASE, a tak dostávame:

```
SELECT
  CASE
    WHEN poradie_od = poradie_do THEN
      RTRIM(CHAR(poradie_od)) || '.'
    ELSE
      RTRIM(CHAR(poradie_od)) || '.' || ' - ' || RTRIM(CHAR(poradie_do)) || '.'
  END
  || ' ' || meno || ' ' || priezvisko AS študent
FROM
(
  SELECT
    š.meno,
    š.priezvisko,
    (
      SELECT COUNT(*) + 1
      FROM študent
      WHERE priemer < š.priemer
    ) AS poradie_od,
    (
      SELECT COUNT(*)
      FROM študent
      WHERE priemer <= š.priemer
    ) AS poradie_do
  FROM študent AS š
) AS pomocná
ORDER BY poradie_od
```

Odpoveď:

ŠTUDENT
1. Ferdinand Mravec
2. Ružena Šípová
3. Dana Botková
4. Jana Botková
5. Mikuláš Myšiak
6. - 7. Ján Hraško
6. - 7. Donald Káčer
8. Aladár Baba
9. Aladár Miazga
10. Ján Polienko
11. Jozef Námorník
12. - 13. Juraj Truľo
12. - 13. Ján Hlúpy

1.6 Manipulácia s dátami tabuľky

Ak nerátame prvotné naplnenie tabuľky, doteraz sme pracovali len s už existujúcimi dátami a obsah tabuľky sme vôbec nemenili. Keďže však máloktorý údaj má trvalú platnosť, musíme mať aj prostriedky, ktoré obsah tabuľky dokážu modifikovať. Za základné operácie možno považovať vkladanie a odstraňovanie záznamov. Hoci ďalšiu operáciu – modifikovanie záznamov – z nich možno zložiť, zaslúži si tiež osobitnú pozornosť.

1.6.1 Vkladanie záznamov

Predstavme si, že na univerzitu nastúpi nový študent – prvák Peter Pan, narodený 13.1.2001, so zatiaľ neznámym priemerom. Už poznáme príkaz na vkladanie záznamu (prípadne viacerých):

```
INSERT INTO študent
VALUES ('Peter', 'Pan', 'muž', '13.1.2001', 1, NULL)
```

Dodajme, že ekvivalentne by poslúžil príkaz s inými formátmi dátumu:

```
INSERT INTO študent
VALUES ('Peter', 'Pan', 'muž', '2001-01-13', 1, NULL)
```

resp.:

```
INSERT INTO študent
VALUES ('Peter', 'Pan', 'muž', '01/13/2001', 1, NULL)
```

Tento príkaz silno využíva znalosť poradia stĺpcov. Ak poradie nepoznáme, môžeme ekvivalentne napísať (pravdaže, poradie stĺpca a hodnoty, ktorú doň vkladáme, musí byť rovnaké):

```
INSERT INTO študent (priezvisko, meno, ročník, dátum_narodenia, pohlavie, priemer)
VALUES ('Pan', 'Peter', 1, '13.1.2001', 'muž', NULL)
```

Vzhľadom na to, že nepoznáme Panov priemer, možno príslušný stĺpec ekvivalentne vynechať:

```
INSERT INTO študent (priezvisko, meno, pohlavie, ročník, dátum_narodenia)
VALUES ('Pan', 'Peter', 'muž', 1, '13.1.2001')
```

Po aplikovaní ktoréhokoľvek z týchto príkazov bude tabuľka vyzerať takto:

```
SELECT *
FROM študent
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ružena	Šípová	žena	1984-02-01	1	1,22
Aladár	Baba	muž	1980-01-22	2	2,03
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Ján	Polienko	muž	1982-04-14	5	2,28
Juraj	Truľo	muž	1979-07-16	1	3,00
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40
Ján	Hlúpy	muž	NULL	2	3,00
Aladár	Miazga	muž	1987-12-22	3	2,06
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Jozef	Námorník	muž	1981-09-23	2	2,90
Peter	Pan	muž	2001-01-13	1	NULL

Je tam!

Do tabuľky môžeme skopírovaním pridať aj záznamy z inej tabuľky spĺňajúce istú podmienku (pravdaže, musia mať správne dátové typy), a to bez ohľadu na to, koľko záznamov v tej druhej tabuľke je. Predstavme si, že banícka fakulta akejsi cudzej univerzity má byť zrušená a jej študenti majú prejsť na našu. Noví študenti sú zatiaľ v tabuľke `študent_baník` s rovnakou štruktúrou. Všimnime si tu novú konštrukciu tabuľky pomocou `LIKE` („ako“):

```
CREATE TABLE študent_baník LIKE študent
```

a s dátami:

```
INSERT INTO študent_baník
VALUES
  ('Grumpy',  'Dwarf', 'muž',  '20.1.1157',  5, 2.23),
  ('Doc',     'Dwarf', 'muž',  '11.5.1254',  4, 1.00),
  ('Bashful', 'Dwarf', 'muž',  '12.6.1197',  4, 2.02),
  ('Sleepy',  'Dwarf', 'muž',  '2.12.1292',  1, 4.50),
  ('Sneezy',  'Dwarf', 'muž',  '13.7.1251',  2, 4.28),
  ('Happy',   'Dwarf', 'muž',  '11.2.1173',  3, 1.00),
  ('Dopey',   'Dwarf', 'muž',  '15.10.1390', 4, 1.50),
  ('Snow',    'White', 'žena', '1.1.1967',   1, 1.00)
```

Stav tabuľky `študent_baník` je teda takýto:

```
SELECT *
FROM študent_baník
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Grumpy	Dwarf	muž	1157-01-20	5	2,23
Doc	Dwarf	muž	1254-05-11	4	1,00
Bashful	Dwarf	muž	1197-06-12	4	2,02
Sleepy	Dwarf	muž	1292-12-02	1	4,50
Sneezy	Dwarf	muž	1251-07-13	2	4,28
Happy	Dwarf	muž	1173-02-11	3	1,00
Dopey	Dwarf	muž	1390-10-15	4	1,50
Snow	White	žena	1967-01-01	1	1,00

Všetky tieto údaje skopírujeme do tabuľky `študent` najjednoduchšie takto:

```
INSERT INTO študent
SELECT *
FROM študent_baník
```

Tento zápis dáva zmysel, ak si spomenieme, že aj na výpis údajov máme dve možnosti – explicitné pomocou `VALUES`, implicitné príkazom `SELECT`. Aj tu sú teda po prvej časti príkazu `INSERT` prípustné obe pokračovania.

Stav tabuľky `študent_baník` sa po tomto príkaze nezmení, no do tabuľky `študent` pribudnú nové záznamy:

```
SELECT *
FROM študent
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ružena	Šípová	žena	1984-02-01	1	1,22
Aladár	Baba	muž	1980-01-22	2	2,03
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Ján	Polienko	muž	1982-04-14	5	2,28
Juraj	Truľo	muž	1979-07-16	1	3,00
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40
Ján	Hlúpy	muž	NULL	2	3,00
Aladár	Miazga	muž	1987-12-22	3	2,06
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Jozef	Námorník	muž	1981-09-23	2	2,90
Peter	Pan	muž	2001-01-13	1	NULL
Grumpy	Dwarf	muž	1157-01-20	5	2,23
Doc	Dwarf	muž	1254-05-11	4	1,00
Bashful	Dwarf	muž	1197-06-12	4	2,02
Sleepy	Dwarf	muž	1292-12-02	1	4,50
Sneezy	Dwarf	muž	1251-07-13	2	4,28
Happy	Dwarf	muž	1173-02-11	3	1,00
Dopey	Dwarf	muž	1390-10-15	4	1,50
Snow	White	žena	1967-01-01	1	1,00

V prípade, že by v definícii štruktúry tabuľky `študent_baník` bolo prehodené poradie napr. stĺpcov `ročník` a `dátum_narodenia`, museli by sme použiť zložitejšiu, ale ekvivalentnú verziu príkazu:

```
INSERT INTO študent (meno, priezvisko, pohlavie, ročník, dátum_narodenia, priemer)
SELECT *
FROM študent_baník
```

alebo opačne:

```
INSERT INTO študent
SELECT
  meno,
  priezvisko,
  pohlavie,
  dátum_narodenia,
  ročník,
  priemer
FROM študent_baník
```

Toto všetko je pravda len v prípade, ak chceme skopírovať všetky dáta z inej tabuľky. Ak by napríklad Snow White odmietla prestúpiť na našu univerzitu, príkaz by bol v časti `SELECT` doplnený podmienkou:

```
INSERT INTO študent
SELECT *
FROM študent_baník
WHERE NOT (meno = 'Snow' AND priezvisko = 'White')
```

Kvôli jednoduchosti však ďalej predpokladajme, že Dwarfovci ju predsa len presvedčili, aby zostala s nimi.

1.6.2 Úprava záznamov

Keď sa o príchode nových študentov dozvedel roduverný kráľ, prvé, čo ho zaujímalo, bolo poslovenčenie ich mien. Najhoršie dopadla Snow White. Kráľ si spomenul na svoje chabé základy angličtiny a preložil len jej krstné meno. Musela sa teda podriadiť príkazu:

```
UPDATE študent
SET
  meno = 'Sneh',
  priezvisko = 'Ulienka'
WHERE
      meno = 'Snow'
  AND priezvisko = 'White'
```

Slovo UPDATE znamená „aktualizovať“ a slovo SET „nastaviť“, teda celý príkaz možno preložiť „aktualizuj (tabuľku) študent (takto): nastav v stĺpci meno hodnotu 'Sneh', v stĺpci priezvisko hodnotu 'Ulienka' (vo všetkých tých riadkoch), kde (platí podmienka) meno = 'Snow' AND priezvisko = 'White'“. Všimnime si, že stĺpce s hodnotami v sekcii SET sú oddelené čiarkami. Ekvivalentne však môžeme napísať:

```
UPDATE študent
SET (meno, priezvisko) = ('Sneh', 'Ulienka')
WHERE
      meno = 'Snow'
  AND priezvisko = 'White'
```

či dokonca:

```
UPDATE študent
SET (meno, priezvisko) = ('Sneh', 'Ulienka')
WHERE (meno, priezvisko) = ('Snow', 'White')
```

Tabuľka študent sa zmenila takto:

```
SELECT *
FROM študent
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ružena	Šípová	žena	1984-02-01	1	1,22
Aladár	Baba	muž	1980-01-22	2	2,03
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Ján	Polienko	muž	1982-04-14	5	2,28
Juraj	Truľo	muž	1979-07-16	1	3,00
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40
Ján	Hlúpy	muž	NULL	2	3,00
Aladár	Miazga	muž	1987-12-22	3	2,06
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Jozef	Námorník	muž	1981-09-23	2	2,90
Peter	Pan	muž	2001-01-13	1	NULL
Grumpy	Dwarf	muž	1157-01-20	5	2,23
Doc	Dwarf	muž	1254-05-11	4	1,00
Bashful	Dwarf	muž	1197-06-12	4	2,02
Sleepy	Dwarf	muž	1292-12-02	1	4,50
Sneezy	Dwarf	muž	1251-07-13	2	4,28
Happy	Dwarf	muž	1173-02-11	3	1,00
Dopey	Dwarf	muž	1390-10-15	4	1,50
Sneh	Ulienka	žena	1967-01-01	1	1,00

S Dwarfovcami to dopadlo len o niečo lepšie – kráľ si nevedel spomenúť na ich slovenské mená, tak im zmenil len priezvisko. A to všetkým naraz, lebo všetci vyhovovali príslušnej podmienke:

```
UPDATE študent
SET priezvisko = 'Trpaslík'
WHERE priezvisko = 'Dwarf'
```

Tabuľka študent sa zmenila takto:

```
SELECT *
FROM študent
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ružena	Šípová	žena	1984-02-01	1	1,22
Aladár	Baba	muž	1980-01-22	2	2,03
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Ján	Polienko	muž	1982-04-14	5	2,28
Juraj	Truľo	muž	1979-07-16	1	3,00
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40
Ján	Hlúpy	muž	NULL	2	3,00
Aladár	Miazga	muž	1987-12-22	3	2,06
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Jozef	Námorník	muž	1981-09-23	2	2,90
Peter	Pan	muž	2001-01-13	1	NULL
Grumpy	Trpaslík	muž	1157-01-20	5	2,23
Doc	Trpaslík	muž	1254-05-11	4	1,00
Bashful	Trpaslík	muž	1197-06-12	4	2,02
Sleepy	Trpaslík	muž	1292-12-02	1	4,50
Sneezy	Trpaslík	muž	1251-07-13	2	4,28
Happy	Trpaslík	muž	1173-02-11	3	1,00
Dopey	Trpaslík	muž	1390-10-15	4	1,50
Sneh	Ulienka	žena	1967-01-01	1	1,00

Po protestoch niekoľkých organizácií za ľudské a trpasličie práva však musel kráľ tento svoj exces napraviť:

```
UPDATE študent
SET
  meno = 'Snow',
  priezvisko = 'White'
WHERE
      meno = 'Sneh'
  AND priezvisko = 'Ulienka'
;
UPDATE študent
SET priezvisko = 'Dwarf'
WHERE priezvisko = 'Trpaslík'
```

Všimnime si, že mal šťastie, že medzi pôvodnými študentmi nebola iná Sneh Ulienka alebo nejaký ôsmy Trpaslík; touto spätnou zmenou by boli totiž postihnutí aj oni, a protesty by určite pokračovali.

A vyvstal ďalší problém: Truľo a Hlúpy, ktorí si zakladali na svojom (ne)prospechu, sa nasrdili, že ich niekto predbehol. Na ich šťastie však múdrejší spolužiaci príslušné orgány upozornili, že noví študenti boli hodnotení známkami v inej stupnici, a preto preniesť bez zmeny údaje o priemeroch do databázy bolo trúfalé. Vymysleli aj lineárnu funkciu f, ktorá by predpokladanú cudziu stupnicu 1 až 5 transformovala do ich stupnice 1 až 3: $f(x) = 1 + \frac{x-1}{2}$. Táto funkcia sa potom prejavila v takomto príkaze:

```
UPDATE študent
SET priemer = 1 + (priemer - 1)/2
WHERE priezvisko IN ('Dwarf','White')
```

Všimnime si podmienku vo `WHERE`, bez nej by sme nechtiac upravili priemery aj pôvodným študentom, ktorých sa táto transformácia netýka. Tabuľka `študent` po tejto zmene vyzerá takto:

```
SELECT *
FROM študent
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ružena	Šípová	žena	1984-02-01	1	1,22
Aladár	Baba	muž	1980-01-22	2	2,03
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Ján	Polienko	muž	1982-04-14	5	2,28
Juraj	Truľo	muž	1979-07-16	1	3,00
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40
Ján	Hlúpy	muž	NULL	2	3,00
Aladár	Miazga	muž	1987-12-22	3	2,06
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Jozef	Námorník	muž	1981-09-23	2	2,90
Peter	Pan	muž	2001-01-13	1	NULL
Grumpy	Dwarf	muž	1157-01-20	5	1,61
Doc	Dwarf	muž	1254-05-11	4	1,00
Bashful	Dwarf	muž	1197-06-12	4	1,51
Sleepy	Dwarf	muž	1292-12-02	1	2,75
Sneezy	Dwarf	muž	1251-07-13	2	2,64
Happy	Dwarf	muž	1173-02-11	3	1,00
Dopey	Dwarf	muž	1390-10-15	4	1,25
Snow	White	žena	1967-01-01	1	1,00

Vidíme teda, že údaje môžeme aktualizovať nielen na konštantu, ale i výrazom obsahujúcim premenné.

Domnienka, že ide o priemer známok, však bola nesprávna – šlo o priemer násady krompáča, resp. metly. Keďže žiadna transformácia týchto údajov na študijné priemery by zrejme nepomohla, najjednoduchšie je na takéto údaje zabudnúť:

```
UPDATE študent
SET priemer = NULL
WHERE priezvisko IN ('Dwarf','White')
```

Tabuľka `študent`:

```
SELECT *
FROM študent
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ružena	Šípová	žena	1984-02-01	1	1,22
Aladár	Baba	muž	1980-01-22	2	2,03
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Ján	Polienko	muž	1982-04-14	5	2,28
Juraj	Truľo	muž	1979-07-16	1	3,00
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40
Ján	Hlúpy	muž	NULL	2	3,00
Aladár	Miazga	muž	1987-12-22	3	2,06
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Jozef	Námorník	muž	1981-09-23	2	2,90
Peter	Pan	muž	2001-01-13	1	NULL
Grumpy	Dwarf	muž	1157-01-20	5	NULL
Doc	Dwarf	muž	1254-05-11	4	NULL
Bashful	Dwarf	muž	1197-06-12	4	NULL
Sleepy	Dwarf	muž	1292-12-02	1	NULL
Sneezy	Dwarf	muž	1251-07-13	2	NULL
Happy	Dwarf	muž	1173-02-11	3	NULL
Dopey	Dwarf	muž	1390-10-15	4	NULL
Snow	White	žena	1967-01-01	1	NULL

Nastaviť teda možno i prázdnu hodnotu.

Za zmienku stojí, že (podobne ako pri príkaze INSERT) možno použiť vnútorný dopyt. Využijúc fakt, že množiny priezvisk pôvodných a nových študentov sú disjunktné, môžeme predchádzajúci vzťah ekvivalentne prepísať aj takto:

```
UPDATE študent
SET priemer = NULL
WHERE priezvisko IN
      (
        SELECT priezvisko
        FROM študent_baník
      )
```

Ešte presnejšou alternatívou je príkaz:

```
UPDATE študent
SET priemer = NULL
WHERE (meno, priezvisko) IN
      (
        SELECT meno, priezvisko
        FROM študent_baník
      )
```

lebo pri ňom vyžadujeme iba rôznosť mien a priezvisk zároveň.

Vnútorný dopyt možno použiť aj v časti SET. Ak by sme napríklad chceli Snow White preradiť do ročníka jej starého hollywoodského priateľa Mikuláša Myšiaka, zabral by takýto príkaz:

```
UPDATE študent
SET ročník =
    (
      SELECT ročník
      FROM študent
      WHERE
            meno = 'Mikuláš'
        AND priezvisko = 'Myšiak'
    )
WHERE
      meno = 'Snow'
  AND priezvisko = 'White'
```

Potom totiž dopyt:

```
SELECT *
FROM študent
```

dáva odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ružena	Šípová	žena	1984-02-01	1	1,22
Aladár	Baba	muž	1980-01-22	2	2,03
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Ján	Polienko	muž	1982-04-14	5	2,28
Juraj	Truľo	muž	1979-07-16	1	3,00
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40
Ján	Hlúpy	muž	NULL	2	3,00
Aladár	Miazga	muž	1987-12-22	3	2,06
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Jozef	Námorník	muž	1981-09-23	2	2,90
Peter	Pan	muž	2001-01-13	1	NULL
Grumpy	Dwarf	muž	1157-01-20	5	NULL
Doc	Dwarf	muž	1254-05-11	4	NULL
Bashful	Dwarf	muž	1197-06-12	4	NULL
Sleepy	Dwarf	muž	1292-12-02	1	NULL
Sneezy	Dwarf	muž	1251-07-13	2	NULL
Happy	Dwarf	muž	1173-02-11	3	NULL
Dopey	Dwarf	muž	1390-10-15	4	NULL
Snow	White	žena	1967-01-01	5	NULL

1.6.3 Mazanie záznamov

Po čase sa ukázalo, že zrušenie baníckej fakulty bolo len fámou, a tamojší študenti sa radšej rozhodli vrátiť sa. Pre databázu našej univerzity to znamená vymazanie príslušných údajov z tabuľky. Použijeme na to príkaz DELETE („vymaž“):

```
DELETE FROM študent
WHERE priezvisko IN ('Dwarf','White')
```

Možno to preložiť „vymaž z (tabuľky) `študent` (všetky záznamy), ktoré spĺňajú podmienku `priezvisko IN ('Dwarf','White')`“. Výsledná tabuľka `študent` opäť vyzerá:

```
SELECT *
FROM študent
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ružena	Šípová	žena	1984-02-01	1	1,22
Aladár	Baba	muž	1980-01-22	2	2,03
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Ján	Polienko	muž	1982-04-14	5	2,28
Juraj	Truľo	muž	1979-07-16	1	3,00
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40
Ján	Hlúpy	muž	NULL	2	3,00
Aladár	Miazga	muž	1987-12-22	3	2,06
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Jozef	Námorník	muž	1981-09-23	2	2,90
Peter	Pan	muž	2001-01-13	1	NULL

Aj v príkazoch mazania možno použiť vnútorný dopyt, takže tu sme mohli napísať ekvivalentný (to, pravdaže, opäť iba za predpokladu disjunktnosti celých mien našich a cudzích študentov) príkaz:

```
DELETE FROM študent
WHERE (meno, priezvisko) IN
      (
        SELECT meno, priezvisko
        FROM študent_baník
      )
```

1.6.4 Preddefinovaná hodnota

Na záver tejto podkapitoly prehoďme pár slov o jednom malom zjednodušení. Ak pri vkladaní nového záznamu zvykne mať niektorá jeho položka istú hodnotu, nemusíme ju opakovať pri každom zázname, stačí, že ju uvedieme už pri definovaní tabuľky. V definícii príslušného stĺpca vtedy napíšeme slovné spojenie WITH DEFAULT („s trvajúcim, (a preto) zanedbaným“) alebo ekvivalentne len slovo DEFAULT nasledované príslušnou hodnotou.

Vzhľadom na to, že väčšina prichádzajúcich študentov nastupuje do prvého ročníka, môžeme našu tabuľku definovať takto:

```
CREATE TABLE študent_d1
(
  meno            VARCHAR(10),
  priezvisko      VARCHAR(15),
  pohlavie        CHAR(4),
  dátum_narodenia DATE,
  ročník          INT DEFAULT 1,
  priemer         DEC(3,2)
)
```

Po vložení záznamu s neúplnými dátami:

```
INSERT INTO študent_d1 (priezvisko, meno, pohlavie, dátum_narodenia)
VALUES ('Pan', 'Peter', 'muž', '13.1.2001')
```

resp. ekvivalentne:

```
INSERT INTO študent_d1 (priezvisko, meno, pohlavie, dátum_narodenia, ročník)
VALUES ('Pan', 'Peter', 'muž', '13.1.2001', DEFAULT)
```

(t. j. konkrétnu hodnotu na príslušnom mieste nahradíme slovom DEFAULT) je obsah tabuľky takýto:

```
SELECT *
FROM študent_d1
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Peter	Pan	muž	2001-01-13	1	NULL

Vidíme, že v stĺpci ročník figuruje hodnota 1 napriek tomu, že pri vkladaní sme ju explicitne neuviedli. V tejto súvislosti je zaujímavé si všimnúť, že v stĺpci priemer, ktorý sme tiež neuviedli, je prázdna hodnota – jemu sme totiž pri definícii tabuľky žiadnu hodnotu nepreddefinovali.

Preddefinovanú hodnotu môžeme jednoducho potlačiť tým, že inú hodnotu uvedieme explicitne. Napríklad po vložení piataka:

```
INSERT INTO študent_d1
VALUES ('Ján', 'Polienko', 'muž', '14.4.1982', 5, 2.28)
```

sa náš DEFAULT vôbec neprejaví:

```
SELECT *
FROM student_d1
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Peter	Pan	muž	2001-01-13	1	NULL
Ján	Polienko	muž	1982-04-14	5	2,28

Preddefinovaná hodnota zafunguje aj pri aktualizovaní dát, keď ju (opäť slovom DEFAULT) uvedieme namiesto konkrétnej hodnoty. Napríklad po príkaze:

```
UPDATE student_d1
SET ročník = DEFAULT
WHERE priezvisko = 'Polienko'
```

tabuľka vyzerá takto:

```
SELECT *
FROM student_d1
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Peter	Pan	muž	2001-01-13	1	NULL
Ján	Polienko	muž	1982-04-14	1	2,28

Ak by sme chceli, aby prijímaní študenti boli prijímaní do 0. ročníka, stačí v predošlom príkaze vymeniť 1 za 0. Je tu však aj iná možnosť využívajúca fakt, že stĺpec typu INT má **neutrálnu hodnotu** práve 0:

```
CREATE TABLE student_d0
(
  meno            VARCHAR(10),
  priezvisko      VARCHAR(15),
  pohlavie        CHAR(4),
  dátum_narodenia DATE,
  ročník          INT DEFAULT,
  priemer         DEC(3,2)
)
```

teda za slovom DEFAULT sa 0 písať nemusí.

Po vložení záznamu s neúplnými dátami:

```
INSERT INTO student_d0 (priezvisko, meno, pohlavie, dátum_narodenia, ročník)
VALUES ('Pan', 'Peter', 'muž', '13.1.2001', DEFAULT)
```

je obsah tabuľky takýto:

```
SELECT *
FROM student_d0
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Peter	Pan	muž	2001-01-13	0	NULL

Pre zaujímavosť si všimnime, že aj pre ďalšie číselné dátové typy je neutrálna hodnota DEFAULT 0, pre reťazce '' (prázdny reťazec) a pre dátum je to aktuálny deň (hodnota funkcie CURRENT DATE („(práve) bežiaci dátum“)) (hoci v helpe DB2 sa uvádza, že pri aktualizovaní má dátum DEFAULT hodnotu 0001-01-01).

Keď sme sa už zmienili o `CURRENT DATE`, dodajme, že môže figurovať ako špeciálna (lebo nekonštantná) preddefinovaná hodnota priamo v definícii stĺpca (isteže typu `DATE`).

1.6.Ú Úlohy

1 Aký (jeden) príkaz mal použiť kráľ, aby zmenil nielen priezviská, ale i mená Dwarfovcov, ak by poznal ich slovenské náprotivky?

1.7 Primárny kľúč a ďalšie integritné obmedzenia

1.7.1 Primárny kľúč

Verte-neverte, na univerzite sa v tom istom ročníku objavil ďalší Ján Hlúpy a čírou náhodou tiež nevie svoj dátum narodenia, ba dokonca majú obaja rovnaký študijný priemer (prosto, jeden Hlúpejší ako druhý...). Neostáva nič iné, len ho vložiť do tabuľky:

```
INSERT INTO student
VALUES ('Ján', 'Hlúpy', 'muž', NULL, 2, 3.00)
```

Tabuľka `student`:

```
SELECT *
FROM student
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ružena	Šípová	žena	1984-02-01	1	1,22
Aladár	Baba	muž	1980-01-22	2	2,03
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Ján	Polienko	muž	1982-04-14	5	2,28
Juraj	Truľo	muž	1979-07-16	1	3,00
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40
Ján	Hlúpy	muž	NULL	2	3,00
Aladár	Miazga	muž	1987-12-22	3	2,06
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Jozef	Námorník	muž	1981-09-23	2	2,90
Peter	Pan	muž	2001-01-13	1	NULL
Ján	Hlúpy	muž	NULL	2	3,00

Na univerzite však dlho nevydržal a onedlho ho bolo treba z databázy odstrániť. Ľahká pomoc, mazať už vieme:

```
DELETE FROM student
WHERE
      meno = 'Ján'
  AND priezvisko = 'Hlúpy'
```

Tabuľka `student`:

```
SELECT *
FROM student
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ružena	Šípová	žena	1984-02-01	1	1,22
Aladár	Baba	muž	1980-01-22	2	2,03
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Ján	Polienko	muž	1982-04-14	5	2,28
Juraj	Truľo	muž	1979-07-16	1	3,00
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40
Aladár	Miazga	muž	1987-12-22	3	2,06
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Jozef	Námorník	muž	1981-09-23	2	2,90
Peter	Pan	muž	2001-01-13	1	NULL

Vidíme, že sa stala nemilá vec – systém vymazal aj nič netušiaceho prvého Hlúpeho Jana. Povedali sme síce, že jeden je hlúpejší ako druhý, ale ako má databáza rozoznať, ktorý je ten prvý a ktorý ten druhý? Keďže všetky položky oboch záznamov sú rovnaké, tieto záznamy sa nedajú rozlíšiť.

Prirodzenou myšlienkou, ako záznamy odlíšiť, je očíslovať ich – potom každý identifikujeme jeho číslom, t. j. **identifikátorom**. To však znamená, že musíme zmeniť štruktúru tabuľky – pridať jeden stĺpec s obvyklým názvom `id`. (Môžeme to urobiť viacerými spôsobmi, no teraz sa upriamime na ten najjednoduchší – tabuľku zmažeme a vytvoríme ju nanovo. Treba však povedať, že tento spôsob má jednu veľkú nevýhodu – stratia sa pri ňom z databázy všetky dáta.) Na odstránenie tabuľky slúži príkaz `DROP TABLE` („strať tabuľku“):

```
DROP TABLE študent
```

V tomto okamihu tabuľka `študent` neexistuje a otázka na jej obsah by dopadla zle. Vytvorme ju teda opäť, už s pridaným stĺpcom:

```
CREATE TABLE študent
(
  id              INT,
  meno            VARCHAR(10),
  priezvisko      VARCHAR(15),
  pohlavie        CHAR(4),
  dátum_narodenia DATE,
  ročník          INT,
  priemer         DEC(3,2)
)
```

a:

```
INSERT INTO študent
VALUES
  (1,  'Ján',       'Hraško',   'muž',  '12.7.1987',  1, 1.83),
  (2,  'Ružena',    'Šípová',   'žena', '1.2.1984',   1, 1.22),
  (3,  'Aladár',    'Baba',     'muž',  '22.1.1980',  2, 2.03),
  (4,  'Ferdinand', 'Mravec',   'muž',  '3.3.1984',   3, 1.00),
  (5,  'Ján',       'Polienko', 'muž',  '14.4.1982',  5, 2.28),
  (6,  'Juraj',     'Truľo',    'muž',  '16.7.1979',  1, 3.00),
  (7,  'Jana',      'Botková',  'žena', '21.9.1977',  4, 1.50),
  (8,  'Dana',      'Botková',  'žena', '21.9.1977',  4, 1.40),
  (9,  'Ján',       'Hlúpy',    'muž',  NULL,         2, 3.00),
  (10, 'Aladár',    'Miazga',   'muž',  '22.12.1987', 3, 2.06),
  (11, 'Mikuláš',   'Myšiak',   'muž',  '6.6.1983',   5, 1.66),
  (12, 'Donald',    'Káčer',    'muž',  '7.10.1982',  5, 1.83),
  (13, 'Jozef',     'Námorník', 'muž',  '23.9.1981',  2, 2.90)
```

Tabuľka `študent`:

```
SELECT *
FROM študent
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
1	Ján	Hraško	muž	1987-07-12	1	1,83
2	Ružena	Šípová	žena	1984-02-01	1	1,22
3	Aladár	Baba	muž	1980-01-22	2	2,03
4	Ferdinand	Mravec	muž	1984-03-03	3	1,00
5	Ján	Polienko	muž	1982-04-14	5	2,28
6	Juraj	Truľo	muž	1979-07-16	1	3,00
7	Jana	Botková	žena	1977-09-21	4	1,50
8	Dana	Botková	žena	1977-09-21	4	1,40
9	Ján	Hlúpy	muž	NULL	2	3,00
10	Aladár	Miazga	muž	1987-12-22	3	2,06
11	Mikuláš	Myšiak	muž	1983-06-06	5	1,66
12	Donald	Káčer	muž	1982-10-07	5	1,83
13	Jozef	Námorník	muž	1981-09-23	2	2,90

I keď číslo pridelené študentovi je umelé, keďže s ním ako takým nijako nesúvisí, ľahko ho pomocou neho identifikujeme. Tak môžeme jeho záznam modifikovať, zmazať, či (za predpokladu správneho udelenia nového čísla) vložiť, a to bez strachu, že by sme zasiahli viacero záznamov (ako sa nám to stalo pri oboch Hlúpych Janoch). Je tu však malý zádrheľ – ten predpoklad správneho udelenia nového čísla. Veľmi ľahko sa totiž môže stať, že sa nejaký záznam v tabuľke ocitne dvakrát – zabudne sa, že už tam bol raz vložený (povedzme, že bol pri výpise prehliadnutý – argument „pozriem, a vidím“ pri väčšom množstve záznamov nemusí obstáť), alebo niekto pri jeho vkladaní čosi stlačí omylom dva razy. Po týždni si možno všimneme, že na toto trochu nešikovné preklápanie dát do novej tabuľky doplatil Peter Pan. Vložíme ho preto osobitne, omylom však s už obsadeným číslom (veď kto môže pri týchto šachoch nestratiť prehľad?):

```
INSERT INTO študent
VALUES (13, 'Peter', 'Pan', 'muž', '13.1.2001', 1, NULL)
```

Tabuľka `študent`:

```
SELECT *
FROM študent
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
1	Ján	Hraško	muž	1987-07-12	1	1,83
2	Ružena	Šípová	žena	1984-02-01	1	1,22
3	Aladár	Baba	muž	1980-01-22	2	2,03
4	Ferdinand	Mravec	muž	1984-03-03	3	1,00
5	Ján	Polienko	muž	1982-04-14	5	2,28
6	Juraj	Truľo	muž	1979-07-16	1	3,00
7	Jana	Botková	žena	1977-09-21	4	1,50
8	Dana	Botková	žena	1977-09-21	4	1,40
9	Ján	Hlúpy	muž	NULL	2	3,00
10	Aladár	Miazga	muž	1987-12-22	3	2,06
11	Mikuláš	Myšiak	muž	1983-06-06	5	1,66
12	Donald	Káčer	muž	1982-10-07	5	1,83
13	Jozef	Námorník	muž	1981-09-23	2	2,90
13	Peter	Pan	muž	2001-01-13	1	NULL

A sme tam, kde sme boli – opäť máme problém s identifikovaním človeka. Nie je totiž v ľudských silách zabezpečiť neomylné prideľovanie čísel novým záznamom, skôr či neskôr by takýto zmätok nastal v akokoľvek dobre strážanej tabuľke. Kontrolu rôznosti vkladaných identifikačných čísel treba nechať na databázu. A práve na to slúži takzvaný **primárny kľúč**. Pod **kľúčom** tabuľky vo všeobecnosti rozumieme stĺpec alebo skupinu stĺpcov, ktorých hodnoty jednoznačne určujú, t. j. identifikujú záznam. Takýchto kľúčov môže byť aj viac, no iba jeden z nich (určený spravidla pri definícii tabuľky) je primárny (ostatné potom nazývame **sekundárne**).

V našom prípade kľúčov nie je veľa. Dlhý čas bola dobrým kandidátom dvojica stĺpcov `meno` a `priezvisko`, ale výskyt dvoch ľudí s rovnakým menom i priezviskom (čo je vcelku bežná vec) ukázal, že nevyhovuje. Stĺpec `ročník` je vzhľadom na svoj význam nevhodný (zrejme sa tá istá hodnota bude opakovať vo viacerých záznamoch) a `priemer` ako desatinné číslo nevyhovuje principiálne (a to ani v kombinácii s inými stĺpcami). Jediným rozumným kľúčom je tu stĺpec `id`, takže zároveň musí byť aj primárnym. Treba povedať, že **primárny kľúč je najdôležitejšie integritné obmedzenie, a tabuľky bez neho preto v dobre navrhnutej databáze nemajú čo hľadať**!

Syntakticky môžeme v definícii tabuľky vyjadriť primárny kľúč napísaním slovného spojenia `PRIMARY KEY` (v preklade „primárny kľúč“), za ktorým nasleduje zoznam stĺpcov tvoriacich kľúč, za definíciu posledného stĺpca po čiarke. Ak je primárny kľúč tvorený len jedným stĺpcom (čo je obvyklý prípad), môžeme to vyjadriť aj alternatívne – napísaním `PRIMARY KEY` hneď za definíciu tohto stĺpca.

Prvoradou podmienkou príslušnosti stĺpca k primárnemu kľúču je však nepripúšťanie prázdnych hodnôt v tomto stĺpci. Túto vlastnosť však môžu mať aj ďalšie stĺpce, dokonca by mal existovať len veľmi vážny

dôvod, aby ju nemali. V definícii tabuľky to vyjadríme tak, že hneď za meno stĺpca napíšeme NOT NULL („nie prázdne hodnoty“).

Dohodli sme sa už, že v našom prípade bude primárnym kľúčom stĺpec id, z ďalších stĺpcov by mali byť NOT NULL ešte meno a priezvisko, pohlavie, ročník (ak chce byť študent študentom, musí byť v nejakom ročníku) i dátum_narodenia (aj Hlúpy Jano sa niekedy musel narodiť, takže ak chce ďalej študovať, nech si to len pekne zistí!). Výnimkou je priemer, ten hlavne u začínajúcich študentov ešte nemusí byť známy (z čoho by mal byť vypočítaný?). Takže po zrušení tabuľky:

```
DROP TABLE študent
```

definujme:

```
CREATE TABLE študent
(
  id              INT         NOT NULL PRIMARY KEY,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4)     NOT NULL,
  dátum_narodenia DATE        NOT NULL,
  ročník          INT         NOT NULL,
  priemer         DEC(3,2)
)
```

alebo ekvivalentne:

```
CREATE TABLE študent
(
  id              INT         NOT NULL,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4)     NOT NULL,
  dátum_narodenia DATE        NOT NULL,
  ročník          INT         NOT NULL,
  priemer         DEC(3,2),
  PRIMARY KEY (id)
)
```

Ak by (teoreticky) tvorili primárny kľúč napr. stĺpce meno a priezvisko, v predposlednom riadku by namiesto PRIMARY KEY (id) bolo PRIMARY KEY (meno, priezvisko).

Ešte tabuľku naplňme (už aj so správne očíslovaným Petrom Panom):

```
INSERT INTO študent
VALUES
  (1,  'Ján',       'Hraško',   'muž',  '12.7.1987',  1, 1.83),
  (2,  'Ružena',    'Šípová',   'žena', '1.2.1984',   1, 1.22),
  (3,  'Aladár',    'Baba',     'muž',  '22.1.1980',  2, 2.03),
  (4,  'Ferdinand', 'Mravec',   'muž',  '3.3.1984',   3, 1.00),
  (5,  'Ján',       'Polienko', 'muž',  '14.4.1982',  5, 2.28),
  (6,  'Juraj',     'Truľo',    'muž',  '16.7.1979',  1, 3.00),
  (7,  'Jana',      'Botková',  'žena', '21.9.1977',  4, 1.50),
  (8,  'Dana',      'Botková',  'žena', '21.9.1977',  4, 1.40),
  (9,  'Ján',       'Hlúpy',    'muž',  '1.4.1972',   2, 3.00),
  (10, 'Aladár',    'Miazga',   'muž',  '22.12.1987', 3, 2.06),
  (11, 'Mikuláš',   'Myšiak',   'muž',  '6.6.1983',   5, 1.66),
  (12, 'Donald',    'Káčer',    'muž',  '7.10.1982',  5, 1.83),
  (13, 'Jozef',     'Námorník', 'muž',  '23.9.1981',  2, 2.90),
  (14, 'Peter',     'Pan',      'muž',  '13.1.2001',  1, NULL)
```

Stav tabuľky študent:

```
SELECT *
FROM študent
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
1	Ján	Hraško	muž	1987-07-12	1	1,83
2	Ružena	Šípová	žena	1984-02-01	1	1,22
3	Aladár	Baba	muž	1980-01-22	2	2,03
4	Ferdinand	Mravec	muž	1984-03-03	3	1,00
5	Ján	Polienko	muž	1982-04-14	5	2,28
6	Juraj	Truľo	muž	1979-07-16	1	3,00
7	Jana	Botková	žena	1977-09-21	4	1,50
8	Dana	Botková	žena	1977-09-21	4	1,40
9	Ján	Hlúpy	muž	1972-04-01	2	3,00
10	Aladár	Miazga	muž	1987-12-22	3	2,06
11	Mikuláš	Myšiak	muž	1983-06-06	5	1,66
12	Donald	Káčer	muž	1982-10-07	5	1,83
13	Jozef	Námorník	muž	1981-09-23	2	2,90
14	Peter	Pan	muž	2001-01-13	1	NULL

Všimnime si, že Ján Hlúpy si radšej dátum zistil – ak by to neurobil, celý príkaz zlyhá, a do tabuľky sa tak nedostane nielen on, ale ani žiaden z jeho spolužiakov. (Na tomto mieste pripomeňme princíp, že **SQL príkaz sa buď vykoná celý, alebo sa nevykoná vôbec**.) Teraz už do tabuľky nemôžeme ani omylom vložiť niekoho s už obsadeným číslom (databáza by okamžite zahlásila chybu), a to dokonca ani ako súčasť ďalšej dávky. Zlyhá teda nielen príkaz:

```
INSERT INTO študent
VALUES (14, 'Snow', 'White', 'žena', '1.1.1967', 1, 1.00)
```

ale i napr.:

```
INSERT INTO študent
VALUES
  (14, 'Snow',   'White', 'žena', '1.1.1967',  1, 1.00),
  (15, 'Grumpy', 'Dwarf', 'muž',  '20.1.1157', 5, 2.23)
```

hoci študent s číslom 15 ešte v tabuľke nie je. Nepomôže ani takáto obchádzka:

```
INSERT INTO študent
VALUES (15, 'Snow', 'White', 'žena', '1.1.1967', 1, 1.00)
;
UPDATE študent
SET id = 14
WHERE id = 15
```

zlyhá totiž druhý z príkazov. Dostali by sme sa totiž do stavu porušujúceho podmienku jedinečnosti hodnôt primárneho kľúča.

Radšej preto po tomto nepodarenom podfuku hneď zahlaďme stopy:

```
DELETE FROM študent
WHERE id = 15
```

Vo všeobecnosti však meniť hodnotu primárneho kľúča môžeme, samozrejme, nová hodnota nesmie byť obsadená:

```
UPDATE študent
SET id = 100
WHERE id = 1
```

Tabuľka sa zmenila takto:

```
SELECT *
FROM študent
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
100	Ján	Hraško	muž	1987-07-12	1	1,83
2	Ružena	Šípová	žena	1984-02-01	1	1,22
3	Aladár	Baba	muž	1980-01-22	2	2,03
4	Ferdinand	Mravec	muž	1984-03-03	3	1,00
5	Ján	Polienko	muž	1982-04-14	5	2,28
6	Juraj	Truľo	muž	1979-07-16	1	3,00
7	Jana	Botková	žena	1977-09-21	4	1,50
8	Dana	Botková	žena	1977-09-21	4	1,40
9	Ján	Hlúpy	muž	1972-04-01	2	3,00
10	Aladár	Miazga	muž	1987-12-22	3	2,06
11	Mikuláš	Myšiak	muž	1983-06-06	5	1,66
12	Donald	Káčer	muž	1982-10-07	5	1,83
13	Jozef	Námorník	muž	1981-09-23	2	2,90
14	Peter	Pan	muž	2001-01-13	1	NULL

Môžeme preto usúdiť, že **databázový systém kontroluje, či po vykonaní celého príkazu nenastane stav odporujúci niektorej z deklarovaných podmienok**, a to bez ohľadu na to, akým spôsobom by sme tento stav chceli docieliť.

Dodajme ešte jednu užitočnú pomôcku. Aby sme pri vkladaní nového záznamu nemuseli stále hľadať, ktoré číslo ešte nebolo použité, možno použiť vnorený dopyt:

```
INSERT INTO student
VALUES (1 + (SELECT MAX(id) FROM student), 'Grumpy', 'Dwarf', 'muž', '20.1.1157', 5, NULL)
```

Aby však takýto dopyt zafungoval aj pre prípadnú prázdnu tabuľku, treba radšej používať zložitejšiu verziu:

```
INSERT INTO student
VALUES (1 + (SELECT VALUE(MAX(id),0) FROM student), 'Snow', 'White', 'žena', '1.1.1967', 1, NULL)
```

A naozaj, po aplikovaní týchto dvoch príkazov naša tabuľka vyzerá takto:

```
SELECT *
FROM student
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
100	Ján	Hraško	muž	1987-07-12	1	1,83
2	Ružena	Šípová	žena	1984-02-01	1	1,22
3	Aladár	Baba	muž	1980-01-22	2	2,03
4	Ferdinand	Mravec	muž	1984-03-03	3	1,00
5	Ján	Polienko	muž	1982-04-14	5	2,28
6	Juraj	Truľo	muž	1979-07-16	1	3,00
7	Jana	Botková	žena	1977-09-21	4	1,50
8	Dana	Botková	žena	1977-09-21	4	1,40
9	Ján	Hlúpy	muž	1972-04-01	2	3,00
10	Aladár	Miazga	muž	1987-12-22	3	2,06
11	Mikuláš	Myšiak	muž	1983-06-06	5	1,66
12	Donald	Káčer	muž	1982-10-07	5	1,83
13	Jozef	Námorník	muž	1981-09-23	2	2,90
14	Peter	Pan	muž	2001-01-13	1	NULL
101	Grumpy	Dwarf	muž	1157-01-20	5	NULL
102	Snow	White	žena	1967-01-01	1	NULL

Po vyprázdnení tabuľky:

```
DELETE
FROM student
```

prvý, jednoduchší príkaz vloženia zlyhá, no po aplikovaní zložitejšieho dostávame po dopyte:

```
SELECT *
FROM študent
```

odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
1	Snow	White	žena	1967-01-01	1	NULL

1.7.2 Sekundárny kľúč

Pozrime sa teraz na ďalšie integritné obmedzenia (súvisiace s jednou tabuľkou). Spomínali sme už sekundárny kľúč. Ak máme takúto skupinu stĺpcov (najčastejšie jeden), ktorá nemá pripúšťať opakujúce sa hodnoty, môžeme kontrolu tohto obmedzenia ponechať na databázu tým, že pri definícii tabuľky vyhlásime túto skupinu za `UNIQUE` („jednoznačnú“), a to rovnako ako pri `PRIMARY KEY` – za posledný stĺpec v tabuľke napíšeme za slovom `UNIQUE` do zátvoriek zoznam stĺpcov tvoriacich túto skupinu. V prípade, že je jednoznačná skupina tvorená len jedným stĺpcom, máme aj druhú možnosť – kľúčové slovo `UNIQUE` napísať priamo v definícii príslušného stĺpca. Dodajme, že (tak ako v prípade primárneho kľúča) nesmú obsahovať stĺpce z jednoznačnej skupiny prázdne hodnoty (a teda musia byť okrem `UNIQUE` aj `NOT NULL`).

Ak by bola podmienka jednoznačnosti položená na stĺpec `priezvisko`, definícia tabuľky by mohla vyzerať takto (aby boli naše ukážky priezračnejšie, porušme (pozor, len na pokusné účely!) predsavzatie, že každá tabuľka bude mať primárny kľúč):

```
CREATE TABLE študent_u_p
(
  id              INT,
  meno            VARCHAR(10),
  priezvisko      VARCHAR(15) NOT NULL UNIQUE,
  pohlavie        CHAR(4),
  dátum_narodenia DATE,
  ročník          INT,
  priemer         DEC(3,2)
)
```

alebo ekvivalentne:

```
CREATE TABLE študent_u_p
(
  id              INT,
  meno            VARCHAR(10),
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4),
  dátum_narodenia DATE,
  ročník          INT,
  priemer         DEC(3,2),
  UNIQUE (priezvisko)
)
```

V takom prípade by zlyhal druhý z týchto príkazov:

```
INSERT INTO študent_u_p
VALUES (1, 'Jana', 'Botková', 'žena', '21.9.1977', 4, 1.50)
;
INSERT INTO študent_u_p
VALUES (2, 'Dana', 'Botková', 'žena', '21.9.1977', 4, 1.40)
```

pretože by sme sa ním pokúšali vložiť záznam s už existujúcou hodnotou stĺpca, ktorý bol deklarovaný ako jednoznačný. Zlyhala by, samozrejme, aj „obchádzka“ – ak by sme druhý záznam vložili s upraveným menom a potom sa ho snažili spätne zmeniť na žiadané:

```
INSERT INTO študent_u_p
VALUES (2, 'Dana', 'Bodková', 'žena', '21.9.1977', 4, 1.40)
;
UPDATE študent_u_p
SET priezvisko = 'Botková'
WHERE id = 2
```

Vidíme teda, že aj toto integritné obmedzenie sa prejavuje ako pri vkladaní, tak pri modifikovaní záznamov, čo potvrdzuje nedávno vyslovené tvrdenie, že **databáze nezáleží na spôsobe porušenia dohodnutých pravidiel, iba na porušovaní samotnom** (no, skôr na ich neporušovaní).

Vyjadrime teraz požiadavku na nutnosť rôznosti všetkých mien a zároveň na nutnosť rôznosti všetkých priezvisk:

```
CREATE TABLE študent_u_m_p
(
  id              INT,
  meno            VARCHAR(10) NOT NULL UNIQUE,
  priezvisko      VARCHAR(15) NOT NULL UNIQUE,
  pohlavie        CHAR(4),
  dátum_narodenia DATE,
  ročník          INT,
  priemer         DEC(3,2)
)
```

resp. ekvivalentne takto:

```
CREATE TABLE študent_u_m_p
(
  id              INT,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4),
  dátum_narodenia DATE,
  ročník          INT,
  priemer         DEC(3,2),
  UNIQUE (meno),
  UNIQUE (priezvisko)
)
```

Ako vidíme, na rozdiel od primárneho kľúča **môže byť toto integritné obmedzenie v tabuľke použité viackrát**. Pre opakovanú hodnotu stĺpca `priezvisko` potom zlyhá (tak ako pred chvíľou) druhý z príkazov:

```
INSERT INTO študent_u_m_p
VALUES (1, 'Jana', 'Botková', 'žena', '21.9.1977', 4, 1.50)
;
INSERT INTO študent_u_m_p
VALUES (2, 'Dana', 'Botková', 'žena', '21.9.1977', 4, 1.40)
```

a pre opakovanú hodnotu stĺpca `meno` druhý z príkazov:

```
INSERT INTO študent_u_m_p
VALUES (3, 'Ján', 'Hraško',   'muž', '12.7.1987', 1, 1.83)
;
INSERT INTO študent_u_m_p
VALUES (4, 'Ján', 'Polienko', 'muž', '14.4.1982', 5, 2.28)
```

Skúsme teraz vyjadriť požiadavku, aby sa neopakovali žiaci s rovnakým menom aj priezviskom zároveň (ako sa to stalo nedávno s Jánmi Hlúpymi). Keďže toto integritné obmedzenie sa týka viacerých stĺpcov, do úvahy prichádza len jedna z doteraz používaných možností definovania obmedzenia, a to mimo definície stĺpcov. Všimnime si syntax – v rámci zátvorky patriacej k `UNIQUE` je viacero stĺpcov oddelených čiarkami:

```
CREATE TABLE študent_u_mp
(
```

```
  id              INT,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4),
  dátum_narodenia DATE,
  ročník          INT,
  priemer         DEC(3,2),
  UNIQUE (meno, priezvisko)
)
```

Teraz už dopadnú predchádzajúce zlyhania úspešne, pretože v prvom prípade sa záznamy líšia v hodnote stĺpca `meno`:

```
INSERT INTO študent_u_mp
VALUES
  (1, 'Jana', 'Botková', 'žena', '21.9.1977', 4, 1.50),
  (2, 'Dana', 'Botková', 'žena', '21.9.1977', 4, 1.40)
```

a v druhom v hodnote stĺpca `priezvisko`:

```
INSERT INTO študent_u_mp
VALUES
  (3, 'Ján', 'Hraško',   'muž', '12.7.1987', 1, 1.83),
  (4, 'Ján', 'Polienko', 'muž', '14.4.1982', 5, 2.28)
```

Zlyhá však druhý z nasledujúcich príkazov, lebo usporiadaná dvojica hodnôt stĺpcov `meno` a `priezvisko` by sa opakovala:

```
INSERT INTO študent_u_mp
VALUES (5, 'Ján', 'Hlúpy', 'muž', '1.4.1972', 2, 3.00)
;
INSERT INTO študent_u_mp
VALUES (6, 'Ján', 'Hlúpy', 'muž', '1.4.1972', 2, 3.00)
```

1.7.3 Kontrola

Ďalším integritným obmedzením je **kontrola**. Ak vieme, že nejaký stĺpec má obsahovať iba hodnoty patriace do istej množiny konštánt, použijeme naň integritné obmedzenie `CHECK` („kontrola“), za ktoré do zátvoriek napíšeme podmienku, ktorú by mal záznam spĺňať. Táto podmienka môže obsahovať logické spojky, porovnania i funkcie (okrem agregačných), no nie vnorený dopyt. Ak sa teda chceme vyhnúť už raz sa vyskytnuvším problémom s priemerom mimo predpokladaného rozsahu, definícia tabuľky bude vyzerať takto (opäť na chvíľu ignorujme požiadavku na existenciu primárneho kľúča):

```
CREATE TABLE študent_c_p
(
  id              INT,
  meno            VARCHAR(10),
  priezvisko      VARCHAR(15),
  pohlavie        CHAR(4),
  dátum_narodenia DATE,
  ročník          INT,
  priemer         DEC(3,2)    CHECK (priemer BETWEEN 1 AND 3)
)
```

alebo ekvivalentne (rozdiel je tu vlastne iba v jednej čiarke):

```
CREATE TABLE študent_c_p
(
  id              INT,
  meno            VARCHAR(10),
  priezvisko      VARCHAR(15),
  pohlavie        CHAR(4),
```

```
  dátum_narodenia DATE,
  ročník          INT,
  priemer         DEC(3,2),
  CHECK (priemer BETWEEN 1 AND 3)
)
```

Potom zlyhá vloženie nového záznamu:

```
INSERT INTO študent_c_p
VALUES (1, 'Sleepy', 'Dwarf', 'muž', '2.12.1292', 1, 4.50)
```

ale i modifikácia už existujúceho:

```
INSERT INTO študent_c_p
VALUES (1, 'Sleepy', 'Dwarf', 'muž', '2.12.1292', 1, 2.50)
;
UPDATE študent_c_p
SET priemer = 4.50
WHERE id = 1
```

lebo hodnota `4.50` nespĺňa podmienku uvedenú v kontrole. Je zaujímavé, že napriek takejto podmienke nezlyhá príkaz:

```
INSERT INTO študent_c_p
VALUES (2, 'Peter', 'Pan', 'muž', '13.1.2001', 1, NULL)
```

ani dvojica:

```
INSERT INTO študent_c_p
VALUES (3, 'Ján', 'Hraško', 'muž', '12.7.1987', 1, 1.83)
;
UPDATE študent_c_p
SET priemer = NULL
WHERE id = 3
```

Prázdna hodnota totiž síce podmienke nevyhovuje, ale ju ani (v zmysle už spomínanej trojhodnotovej logiky) neporušuje. **Kontrola teda nestráži splnenie podmienky, iba bráni jej nesplneniu.** Hľa, aký koan!...

Niekedy sa kontrola môže vzťahovať na viacero stĺpcov. Vtedy ju môžeme vyjadriť iba tak, že ju napíšeme až po definovaní všetkých stĺpcov. Ak napríklad chceme za každú cenu zabrániť v štúdiu na našej univerzite Mechúrikovi Koščúrikovi pochybnej povesti, tabuľku definujeme takto:

```
CREATE TABLE študent_c_mp
(
  id              INT,
  meno            VARCHAR(10),
  priezvisko      VARCHAR(15),
  pohlavie        CHAR(4),
  dátum_narodenia DATE,
  ročník          INT,
  priemer         DEC(3,2),
  CHECK (NOT (meno = 'Mechúrik' AND priezvisko = 'Koščúrik'))
)
```

Potom zlyhá aj príkaz vloženia nového:

```
INSERT INTO študent_c_mp
VALUES (1, 'Mechúrik', 'Koščúrik', 'muž', '8.9.1922', 3, 1.11)
```

aj príkaz modifikácie existujúceho záznamu:

```
INSERT INTO študent_c_mp
VALUES (1, 'paMechúrik', 'paKoščúrik', 'muž', '8.9.1922', 3, 1.11)
;
UPDATE študent_c_mp
SET
  meno = 'Mechúrik',
  priezvisko = 'Koščúrik'
WHERE id = 1
```

Podobne ako pri sekundárnych kľúčoch, **tabuľka môže obsahovať aj viacero kontrol**. Ak chceme zabezpečiť obe predchádzajúce podmienky, môžeme ju definovať napríklad takto:

```
CREATE TABLE študent_c_mp_p
(
  id              INT,
  meno            VARCHAR(10),
  priezvisko      VARCHAR(15),
  pohlavie        CHAR(4),
  dátum_narodenia DATE,
  ročník          INT,
  priemer         DEC(3,2),
  CHECK (NOT (meno = 'Mechúrik' AND priezvisko = 'Koščúrik')),
  CHECK (priemer BETWEEN 1 AND 3)
)
```

Výsledná tabuľka potom musí spĺňať každú z nich. Kvôli úplnosti dodajme, že alternatívne môžeme podmienky konjunkciou zlúčiť do jednej:

```
CREATE TABLE študent_c_mp_p
(
  id              INT,
  meno            VARCHAR(10),
  priezvisko      VARCHAR(15),
  pohlavie        CHAR(4),
  dátum_narodenia DATE,
  ročník          INT,
  priemer         DEC(3,2),
  CHECK (NOT (meno = 'Mechúrik' AND priezvisko = 'Koščúrik') AND (priemer BETWEEN 1 AND 3))
)
```

Takéto vyjadrenie má rovnaký efekt, ale viacero nevýhod – napr. pri jeho porušení je chybové hlásenie menej informatívne, pretože potom nevieme rozoznať, ktorá z častí konjunkcie vlastne bola porušená.

2
Práca s viacerými tabuľkami

2.1 Spojenie dvoch tabuliek

2.1.1 Jedna tabuľka nestačí

Náhle vznikla potreba vedieť, odkiaľ tí študenti vôbec sú (uspokojili by sme sa aspoň s krajinou, ale uvítame aj informáciu o jej hlave), pričom budeme prepokladať, že ide o trvalé bydlisko, ktoré je z princípu jediné. V našej tabuľke však miesto pre takúto informáciu nie je – neboli sme vo fáze analýzy dostatočne prezieraví. Neostáva nič iné, len zmeniť štruktúru tabuľky – pridať stĺpec `bydlisko`. Najjednoduchšie (hoci, ako sme už spomínali, nie najšťastnejšie) bude starú tabuľku vymazať a vytvoriť novú, už so správnou štruktúrou:

```
DROP TABLE študent
;
CREATE TABLE študent
(
  id              INT         NOT NULL PRIMARY KEY,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4)     NOT NULL,
  dátum_narodenia DATE        NOT NULL,
  ročník          INT         NOT NULL,
  priemer         DEC(3,2),
  bydlisko        VARCHAR(50),
  hlava_krajiny   VARCHAR(50)
)
```

Keďže u niektorých študentov je pôvod nejasný, stĺpcom `krajina` a `hlava_krajiny` sme povolili prázdne hodnoty.

```
INSERT INTO študent
VALUES
(1, 'Ján',      'Hraško',  'muž', '12.7.1987', 1,1.83,'Za siedmimi horami a siedmimi dolami', 'Drozdia Brada' ),
(2, 'Ružena',   'Šípová',  'žena','1.2.1984',  1,1.22,'Za 7 horami a 7 dolami',               'Dr. Brada'     ),
(3, 'Aladár',   'Baba',    'muž', '22.1.1980', 2,2.03,'Kalifát Bagdad',                       'Harún al-Rašíd'),
(4, 'Ferdinand','Mravec',  'muž', '3.3.1984',  3,1.00,'Mravenisko',                           'Z'             ),
(5, 'Ján',      'Polienko','muž', '14.4.1982', 5,2.28,'Za siedmymi horami a siedmymi dolami', 'Matej Korvín  '),
(6, 'Juraj',    'Truľo',   'muž', '16.7.1979', 1,3.00,'Za siedmimi horami a siedmimi  dolami','Drozdia  Brada'),
(7, 'Jana',     'Botková', 'žena','21.9.1977', 4,1.50,NULL,                                   NULL            ),
(8, 'Dana',     'Botková', 'žena','21.9.1977', 4,1.40,NULL,                                   NULL            ),
(9, 'Ján',      'Hlúpy',   'muž', '1.4.1972',  2,3.00,'ZA SIEDMIMI HORAMI A SIEDMIMI DOLAMI', 'DROZDIA BRADA' ),
(10,'Aladár',   'Miazga',  'muž', '22.12.1987',3,2.06,NULL,                                  NULL            ),
(11,'Mikuláš',  'Myšiak',  'muž', '6.6.1983',  5,1.66,'Hollywood',                           'Simba'         ),
(12,'Donald',   'Káčer',   'muž', '7.10.1982', 5,1.83,'Holy Wood',                           'Simba'         ),
(13,'Jozef',    'Námorník','muž', '23.9.1981', 2,2.90,NULL,                                  NULL            ),
(14,'Peter',    'Pan',     'muž', '13.1.2001', 1,NULL,'Neverland',                           NULL            )
```

Všimnime si však ten zmätok – jedna a tá istá krajina má hneď niekoľko rôznych pomenovaní – niekde sú napríklad použité skratky, niekde číslice, inde je zas nenápadná medzera navyše. Táto situácia sa mohla prihodiť veľmi jednoducho – človek, ktorý mal na starosti zapisovanie nových študentov, možno vybavoval každého v iný deň, raz sa mu nechcelo písať, raz mal zle prepnutú klávesnicu, inokedy ho niekto zastupoval, občas sa prejavili aj jeho nedostatky v zemepise (keď cezmínu považoval za svätú) či v pravopise (skloňovanie číslovky „sedem“).

Ľudská senzitívnosť na syntaktickú rovnosť reťazcov je pomerne nízka. Často sa mylne stotožňujú slová (či „slová“) s rovnakou alebo podobnou výslovnosťou (o tomto fakte svedčí aj existencia (a potreba) školských diktátov). Omnoho vážnejším **dôvodom tejto necitlivosti je však paradoxne naša inteligencia, ktorá nevníma slová ako také, ale zaujíma sa o ich význam, a preto často automaticky ignoruje ich formálne odlišnosti**. (Nestalo sa vám už, že ste si až kdesi v polovici knihy uvedomili, že je v češtine?) Databázový stroj je však zameraný práve na formu, obsah je preň irelevantný. Rozdiel vo vnímaní reťazcov človeka a stroja je tak priveľký, ba až nebezpečný. Preto sa na identifikáciu objektov lepšie hodia **čísla**, ktoré **sú samy sebe významom** a nemajú žiadne zavádzajúce konotácie. (Kto neverí, nech si porovná malý/drobný rozdiel medzi chápaním slov `malý` a `drobný` s diametrálne odlišným vnímaním súm `100` Sk a `1000` Sk líšiacich sa v jedinej

nule. Alebo ak oslovíte svojho priateľa Janíček namiesto Janíčko, ten si to možno ani nevšimne, no skúste sa pri vytáčaní jeho telefónneho čísla pomýliť čo len v jednej cifre. . .)

Vráťme sa k situácii v našej tabuľke. Zadávateľovi údajov by sme mohli pripísať aj ďalšie nezrovnalosti, napríklad v informáciách o hlave štátu Za siedmimi horami a siedmimi dolami, ale tie môžu mať vážnejšiu príčinu: V čase zapisovania Jána Polienka bol kráľom Matej Korvín, a hoci sa odvtedy situácia zmenila a na trón nastúpil kráľ Drozdia Brada, nikto tento údaj v databáze neaktualizoval.

Takto uložené informácie o krajine síce nie sú úplne nepoužiteľné, ale urobiť z nich sumár je v podstate nemožné. Napríklad ak chceme zistiť počet ľudí z každej krajiny, mal by zafungovať dopyt:

```
SELECT
  bydlisko,
  COUNT(*) AS počet_študentov
FROM študent
GROUP BY bydlisko
```

Odpoveď:

BYDLISKO	POČET_ŠTUDENTOV
Hollywood	1
Holy Wood	1
Kalifát Bagdad	1
Mravenisko	1
Neverland	1
Za 7 horami a 7 dolami	1
Za siedmimi horami a siedmimi dolami	1
Za siedmimi horami a siedmimi dolami	1
ZA SIEDMIMI HORAMI A SIEDMIMI DOLAMI	1
Za siedmymi horami a siedmymi dolami	1
NULL	4

Z výpisu vidíme, že prvé dva, resp. predposledných päť riadkov asi má byť spolu (i keď tieň pochybnosti ostáva), ale ťažko to čakať od databázy, neznalej kontextu. Chyba je, samozrejme, v nás – umožnili sme nášmu od prírody neporiadnemu živočíšnemu druhu byť pri tejto činnosti príliš tvorivý.

Na spochybnenie doterajšieho postupu však máme ešte jeden dôvod a ten je omnoho vážnejší. Predpokladajme preto, že akýmsi zázrakom sa údaje o krajinách dostali tak, že v nich nie sú žiadne spomínané formálne prehrešky:

```
DELETE FROM študent
;
INSERT INTO študent
VALUES
(1, 'Ján',      'Hraško',   'muž', '12.7.1987', 1,1.83,'Za siedmimi horami a siedmimi dolami', 'Drozdia Brada' ),
(2, 'Ružena',   'Šípová',   'žena','1.2.1984',  1,1.22,'Za siedmimi horami a siedmimi dolami', 'Drozdia Brada' ),
(3, 'Aladár',   'Baba',     'muž', '22.1.1980', 2,2.03,'Kalifát Bagdad',                       'Harún al-Rašíd'),
(4, 'Ferdinand','Mravec',   'muž', '3.3.1984',  3,1.00,'Mravenisko',                           'Z'             ),
(5, 'Ján',      'Polienko','muž', '14.4.1982', 5,2.28,'Za siedmimi horami a siedmimi dolami', 'Drozdia Brada' ),
(6, 'Juraj',    'Truľo',    'muž', '16.7.1979', 1,3.00,'Za siedmimi horami a siedmimi dolami', 'Drozdia Brada' ),
(7, 'Jana',     'Botková', 'žena','21.9.1977', 4,1.50,NULL,                                   NULL            ),
(8, 'Dana',     'Botková', 'žena','21.9.1977', 4,1.40,NULL,                                   NULL            ),
(9, 'Ján',      'Hlúpy',    'muž', '1.4.1972',  2,3.00,'Za siedmimi horami a siedmimi dolami', 'Drozdia Brada' ),
(10,'Aladár',   'Miazga',   'muž', '22.12.1987',3,2.06,NULL,                                  NULL            ),
(11,'Mikuláš',  'Myšiak',   'muž', '6.6.1983',  5,1.66,'Hollywood',                           'Simba'         ),
(12,'Donald',   'Káčer',    'muž', '7.10.1982', 5,1.83,'Hollywood',                           'Simba'         ),
(13,'Jozef',    'Námorník','muž', '23.9.1981', 2,2.90,NULL,                                  NULL            ),
(14,'Peter',    'Pan',      'muž', '13.1.2001', 1,NULL,'Neverland',                           NULL            )
```

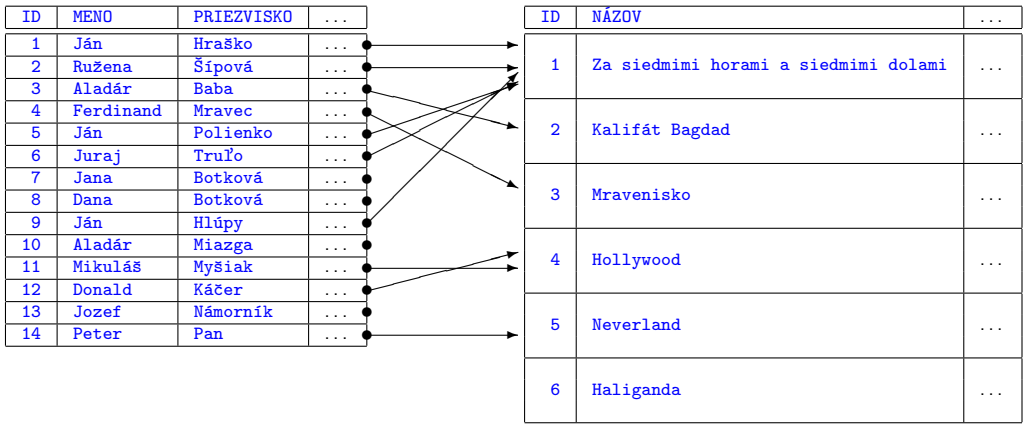
Keď si spomenieme na jeden z hlavných princípov databáz – neduplicitu dát, uvidíme, že tu čosi nehrá: Informáciu o Simbovi ako kráľovi Hollywoodu máme uloženú dvakrát a o Drozdej Brade ako vládcovi krajiny Za siedmimi horami a siedmimi dolami dokonca štyrikrát. Navyše, ak náhodou Ali Baba odíde zo školy, stratíme informáciu, kto je kalifom Bagdadu. Z tejto hromady pochybností vyvstáva jedna podstatná otázka – prečo

by mal byť vzťah krajiny a jej vládcu uložený v tabuľke `študent`? Tá má predsa, ako už jej názov napovedá, združovať informácie o jednotlivých študentoch. Analogicky by teda mohla údaje o krajinách združovať ďalšia tabuľka s názvom `krajina`, a tak ako pri študentoch, aj v nej bude každej krajine zodpovedať jeden očíslovaný riadok:

```
CREATE TABLE krajina
(
  id              INT         NOT NULL PRIMARY KEY,
  názov           VARCHAR(50) NOT NULL,
  hlava           VARCHAR(50)
)
;
INSERT INTO krajina
VALUES
  (1, 'Za siedmimi horami a siedmimi dolami', 'Drozdia Brada' ),
  (2, 'Kalifát Bagdad',                       'Harún al-Rašíd'),
  (3, 'Mravenisko',                           'Z'             ),
  (4, 'Hollywood',                            'Simba'         ),
  (5, 'Neverland',                            NULL            ),
  (6, 'Haliganda',                            NULL            )
```

Krajina Haliganda nech je zástupcom všetkých ostatných nezaangažovaných krajín, napokon, tento zoznam nemusíme používať len v súvislosti so študentmi. Aj preto prázdnu hodnotu do tabuľky ukladať nebudeme. Tabuľky tohto typu, ktorých dáta sa veľmi nemenia, sa zvyknú nazývať **číselníky**.

Vráťme sa však k tabuľke `študent`. Stĺpce `bydlisko` a `hlava_krajiny` z nej teda môžeme odstrániť, keďže sa, hoci premenované, nachádzajú v novej tabuľke `krajina`. Tým však strácame informáciu o trvalom bydlisku študenta. Tu si ale uvedomme, že nie meno krajiny (a už vôbec nie jej hlava) je vlastnosťou študenta, ale jeho príslušnosť k nej. Tento jeho atribút môžeme reprezentovať ako odkaz na príslušný riadok tabuľky `krajina`:



ID	MENO	PRIEZVISKO	...
1	Ján	Hraško	...
2	Ružena	Šípová	...
3	Aladár	Baba	...
4	Ferdinand	Mravec	...
5	Ján	Polienko	...
6	Juraj	Truľo	...
7	Jana	Botková	...
8	Dana	Botková	...
9	Ján	Hlúpy	...
10	Aladár	Miazga	...
11	Mikuláš	Myšiak	...
12	Donald	Káčer	...
13	Jozef	Námorník	...
14	Peter	Pan	...

ID	NÁZOV	...
1	Za siedmimi horami a siedmimi dolami	...
2	Kalifát Bagdad	...
3	Mravenisko	...
4	Hollywood	...
5	Neverland	...
6	Haliganda	...

Takéto odkazy sa v databázach zvyknú vyjadrovať tak, že namiesto šípok sa uvádzajú identifikátory záznamov odkazovanej tabuľky. Do tabuľky `študent` preto pridáme stĺpec `id_bydlisko` a každá jeho hodnota sa bude rovnať identifikátoru príslušnej krajiny (alebo bude prázdna). Okrem iného to znamená, že dátové typy týchto dvoch stĺpcov sa musia zhodovať. Takže tabuľku nanovo vytvoríme:

```
DROP TABLE študent
;
CREATE TABLE študent
(
  id              INT         NOT NULL PRIMARY KEY,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4)     NOT NULL,
  dátum_narodenia DATE        NOT NULL,
  ročník          INT         NOT NULL,
  priemer         DEC(3,2),
  id_bydlisko     INT
)
;
INSERT INTO študent
VALUES
  (1,  'Ján',       'Hraško',   'muž',  '12.7.1987',  1, 1.83, 1   ),
  (2,  'Ružena',    'Šípová',   'žena', '1.2.1984',   1, 1.22, 1   ),
  (3,  'Aladár',    'Baba',     'muž',  '22.1.1980',  2, 2.03, 2   ),
  (4,  'Ferdinand', 'Mravec',   'muž',  '3.3.1984',   3, 1.00, 3   ),
  (5,  'Ján',       'Polienko', 'muž',  '14.4.1982',  5, 2.28, 1   ),
  (6,  'Juraj',     'Truľo',    'muž',  '16.7.1979',  1, 3.00, 1   ),
  (7,  'Jana',      'Botková',  'žena', '21.9.1977',  4, 1.50, NULL),
  (8,  'Dana',      'Botková',  'žena', '21.9.1977',  4, 1.40, NULL),
  (9,  'Ján',       'Hlúpy',    'muž',  '1.4.1972',   2, 3.00, 1   ),
  (10, 'Aladár',    'Miazga',   'muž',  '22.12.1987', 3, 2.06, NULL),
  (11, 'Mikuláš',   'Myšiak',   'muž',  '6.6.1983',   5, 1.66, 4   ),
  (12, 'Donald',    'Káčer',    'muž',  '7.10.1982',  5, 1.83, 4   ),
  (13, 'Jozef',     'Námorník', 'muž',  '23.9.1981',  2, 2.90, NULL),
  (14, 'Peter',     'Pan',      'muž',  '13.1.2001',  1, NULL, 5   )
```

Dodajme ešte, že medzi identifikačnými číslami študentov a krajín nie je žiaden vzťah, hoci majú rovnaký názov `id`. V skutočnosti sú ich plné, tzv. **kvalifikované** názvy `študent.id` a `krajina.id` (teda v tvare *tabuľka.stĺpec*), takže sa predsa len odlišujú.

Proces vydelenia so študentmi priamo nesúvisiacich údajov o ich bydliskách z tabuľky `študent` do novej tabuľky `krajina` budeme nazývať **normalizácia**. Uvedomme si, že sme ním odstránili všetky spomínané problémy: Pri zápise nových študentov sa tak do databázy ukladajú naozaj iba informácie o študentoch, údaje o krajinách môžu byť uložené inokedy (ale skôr). Starosť o vkladanie informácií teda môže byť rozdelená na dve časti – o číselník krajín sa môže starať niekto úplne iný (napr. správca databázy, ktorý ho naplní hneď pri jej vytvorení) než o vypĺňovanie dát o študentoch (to bude robiť zrejme študijné oddelenie). V prípade zmeny názvu krajiny alebo jej hlavy (či už v prípade omylu, alebo pri reálnej zmene politickej situácie) treba a stačí zasiahnuť na jediné miesto v databáze, takže nekonzistencia dát už nehrozí. Keď nám napríklad z Kalifátu Bagdad odíde aj posledný študent, informácie o tejto krajine vôbec nestratíme, len sa jej číslo prestane vyskytovať v stĺpci `id_bydlisko` tabuľky `študent`.

Normalizácia má však okrem odstránenia spomínaných problémov ešte jeden dôležitý význam. Pomáha nám totiž odlíšiť pravé atribúty objektov (t. j. vlastnosti ich samých) od odkazov (t. j. ich vzťahov k ostatným objektom), uvedomiť si tým vzťahy medzi jednotlivými typmi objektov, a tak lepšie pochopiť, ako sa veci majú.

2.1.2 Spojenie

Ako však teraz postupovať, ak chceme vypísať mená a priezviská študentov a ku každému aj jeho krajinu (pravda, ak je táto informácia dostupná)? Urobme prvý naivný pokus – do časti `FROM`, kde sme doteraz písali stále tabuľku `študent` (napokon, inú sme ani nemali), dáme obe tabuľky:

```
SELECT *
FROM
  študent,
  krajina
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER	ID_BYDLISKO	ID	NÁZOV	HLAVA
1	Ján	Hraško	muž	1987-07-12	1	1,83	1	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
2	Ružena	Šípová	žena	1984-02-01	1	1,22	1	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
3	Aladár	Baba	muž	1980-01-22	2	2,03	2	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
4	Ferdinand	Mravec	muž	1984-03-03	3	1,00	3	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
5	Ján	Polienko	muž	1982-04-14	5	2,28	1	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
6	Juraj	Truľo	muž	1979-07-16	1	3,00	1	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
7	Jana	Botková	žena	1977-09-21	4	1,50	NULL	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
8	Dana	Botková	žena	1977-09-21	4	1,40	NULL	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
9	Ján	Hlúpy	muž	1972-04-01	2	3,00	1	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
10	Aladár	Miazga	muž	1987-12-22	3	2,06	NULL	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
11	Mikuláš	Myšiak	muž	1983-06-06	5	1,66	4	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
12	Donald	Káčer	muž	1982-10-07	5	1,83	4	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
13	Jozef	Námorník	muž	1981-09-23	2	2,90	NULL	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
14	Peter	Pan	muž	2001-01-13	1	NULL	5	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
1	Ján	Hraško	muž	1987-07-12	1	1,83	1	2	Kalifát Bagdad	Harún al-Rašíd
2	Ružena	Šípová	žena	1984-02-01	1	1,22	1	2	Kalifát Bagdad	Harún al-Rašíd
3	Aladár	Baba	muž	1980-01-22	2	2,03	2	2	Kalifát Bagdad	Harún al-Rašíd
4	Ferdinand	Mravec	muž	1984-03-03	3	1,00	3	2	Kalifát Bagdad	Harún al-Rašíd
5	Ján	Polienko	muž	1982-04-14	5	2,28	1	2	Kalifát Bagdad	Harún al-Rašíd
6	Juraj	Truľo	muž	1979-07-16	1	3,00	1	2	Kalifát Bagdad	Harún al-Rašíd
7	Jana	Botková	žena	1977-09-21	4	1,50	NULL	2	Kalifát Bagdad	Harún al-Rašíd
8	Dana	Botková	žena	1977-09-21	4	1,40	NULL	2	Kalifát Bagdad	Harún al-Rašíd
9	Ján	Hlúpy	muž	1972-04-01	2	3,00	1	2	Kalifát Bagdad	Harún al-Rašíd
10	Aladár	Miazga	muž	1987-12-22	3	2,06	NULL	2	Kalifát Bagdad	Harún al-Rašíd
11	Mikuláš	Myšiak	muž	1983-06-06	5	1,66	4	2	Kalifát Bagdad	Harún al-Rašíd
12	Donald	Káčer	muž	1982-10-07	5	1,83	4	2	Kalifát Bagdad	Harún al-Rašíd
13	Jozef	Námorník	muž	1981-09-23	2	2,90	NULL	2	Kalifát Bagdad	Harún al-Rašíd
14	Peter	Pan	muž	2001-01-13	1	NULL	5	2	Kalifát Bagdad	Harún al-Rašíd
1	Ján	Hraško	muž	1987-07-12	1	1,83	1	3	Mravenisko	Z
2	Ružena	Šípová	žena	1984-02-01	1	1,22	1	3	Mravenisko	Z
3	Aladár	Baba	muž	1980-01-22	2	2,03	2	3	Mravenisko	Z
4	Ferdinand	Mravec	muž	1984-03-03	3	1,00	3	3	Mravenisko	Z
5	Ján	Polienko	muž	1982-04-14	5	2,28	1	3	Mravenisko	Z
6	Juraj	Truľo	muž	1979-07-16	1	3,00	1	3	Mravenisko	Z
7	Jana	Botková	žena	1977-09-21	4	1,50	NULL	3	Mravenisko	Z
8	Dana	Botková	žena	1977-09-21	4	1,40	NULL	3	Mravenisko	Z
9	Ján	Hlúpy	muž	1972-04-01	2	3,00	1	3	Mravenisko	Z
10	Aladár	Miazga	muž	1987-12-22	3	2,06	NULL	3	Mravenisko	Z
11	Mikuláš	Myšiak	muž	1983-06-06	5	1,66	4	3	Mravenisko	Z
12	Donald	Káčer	muž	1982-10-07	5	1,83	4	3	Mravenisko	Z
13	Jozef	Námorník	muž	1981-09-23	2	2,90	NULL	3	Mravenisko	Z
14	Peter	Pan	muž	2001-01-13	1	NULL	5	3	Mravenisko	Z
1	Ján	Hraško	muž	1987-07-12	1	1,83	1	4	Hollywood	Simba
2	Ružena	Šípová	žena	1984-02-01	1	1,22	1	4	Hollywood	Simba
3	Aladár	Baba	muž	1980-01-22	2	2,03	2	4	Hollywood	Simba
4	Ferdinand	Mravec	muž	1984-03-03	3	1,00	3	4	Hollywood	Simba
5	Ján	Polienko	muž	1982-04-14	5	2,28	1	4	Hollywood	Simba
6	Juraj	Truľo	muž	1979-07-16	1	3,00	1	4	Hollywood	Simba
7	Jana	Botková	žena	1977-09-21	4	1,50	NULL	4	Hollywood	Simba
8	Dana	Botková	žena	1977-09-21	4	1,40	NULL	4	Hollywood	Simba
9	Ján	Hlúpy	muž	1972-04-01	2	3,00	1	4	Hollywood	Simba
10	Aladár	Miazga	muž	1987-12-22	3	2,06	NULL	4	Hollywood	Simba
11	Mikuláš	Myšiak	muž	1983-06-06	5	1,66	4	4	Hollywood	Simba
12	Donald	Káčer	muž	1982-10-07	5	1,83	4	4	Hollywood	Simba
13	Jozef	Námorník	muž	1981-09-23	2	2,90	NULL	4	Hollywood	Simba
14	Peter	Pan	muž	2001-01-13	1	NULL	5	4	Hollywood	Simba
1	Ján	Hraško	muž	1987-07-12	1	1,83	1	5	Neverland	NULL
2	Ružena	Šípová	žena	1984-02-01	1	1,22	1	5	Neverland	NULL
3	Aladár	Baba	muž	1980-01-22	2	2,03	2	5	Neverland	NULL
4	Ferdinand	Mravec	muž	1984-03-03	3	1,00	3	5	Neverland	NULL
5	Ján	Polienko	muž	1982-04-14	5	2,28	1	5	Neverland	NULL
6	Juraj	Truľo	muž	1979-07-16	1	3,00	1	5	Neverland	NULL
7	Jana	Botková	žena	1977-09-21	4	1,50	NULL	5	Neverland	NULL
8	Dana	Botková	žena	1977-09-21	4	1,40	NULL	5	Neverland	NULL
9	Ján	Hlúpy	muž	1972-04-01	2	3,00	1	5	Neverland	NULL
10	Aladár	Miazga	muž	1987-12-22	3	2,06	NULL	5	Neverland	NULL
11	Mikuláš	Myšiak	muž	1983-06-06	5	1,66	4	5	Neverland	NULL
12	Donald	Káčer	muž	1982-10-07	5	1,83	4	5	Neverland	NULL
13	Jozef	Námorník	muž	1981-09-23	2	2,90	NULL	5	Neverland	NULL
14	Peter	Pan	muž	2001-01-13	1	NULL	5	5	Neverland	NULL
1	Ján	Hraško	muž	1987-07-12	1	1,83	1	6	Haliganda	NULL
2	Ružena	Šípová	žena	1984-02-01	1	1,22	1	6	Haliganda	NULL
3	Aladár	Baba	muž	1980-01-22	2	2,03	2	6	Haliganda	NULL
4	Ferdinand	Mravec	muž	1984-03-03	3	1,00	3	6	Haliganda	NULL
5	Ján	Polienko	muž	1982-04-14	5	2,28	1	6	Haliganda	NULL
6	Juraj	Truľo	muž	1979-07-16	1	3,00	1	6	Haliganda	NULL
7	Jana	Botková	žena	1977-09-21	4	1,50	NULL	6	Haliganda	NULL
8	Dana	Botková	žena	1977-09-21	4	1,40	NULL	6	Haliganda	NULL
9	Ján	Hlúpy	muž	1972-04-01	2	3,00	1	6	Haliganda	NULL
10	Aladár	Miazga	muž	1987-12-22	3	2,06	NULL	6	Haliganda	NULL
11	Mikuláš	Myšiak	muž	1983-06-06	5	1,66	4	6	Haliganda	NULL
12	Donald	Káčer	muž	1982-10-07	5	1,83	4	6	Haliganda	NULL
13	Jozef	Námorník	muž	1981-09-23	2	2,90	NULL	6	Haliganda	NULL
14	Peter	Pan	muž	2001-01-13	1	NULL	5	6	Haliganda	NULL

Nedopadlo to najlepšie – každý študent bol skombinovaný s každou krajinou, výsledkom je teda karteziánsky súčin množín záznamov z oboch tabuliek. Z neho nás však nezaujímajú všetky riadky, ale len tie, v ktorých sa zhodujú hodnoty stĺpcov `id_bydlisko` z tabuľky `študent` a `id` z tabuľky `krajina`; vieme totiž, že prvým ukazujeme na druhý. Doplňme preto náš dopyt príslušnou podmienkou. Všimnime si, že v nej majú stĺpce plné mená (i keď zo syntaktického hľadiska je to nutné iba pri `id`, keďže tento názov sa vyskytuje dvakrát). Takýto dopyt nazývame **spojenie** dvoch tabuliek:

```
SELECT *
FROM
  študent,
  krajina
WHERE študent.id_bydlisko = krajina.id
```

alebo (vzhľadom na jedinečnosť názvu stĺpca `id_bydlisko`) ekvivalentne:

```
SELECT *
FROM
  študent,
  krajina
WHERE id_bydlisko = krajina.id
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER	ID_BYDLISKO	ID	NÁZOV	HLAVA
1	Ján	Hraško	muž	1987-07-12	1	1,83	1	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
2	Ružena	Šípová	žena	1984-02-01	1	1,22	1	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
5	Ján	Polienko	muž	1982-04-14	5	2,28	1	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
6	Juraj	Truľo	muž	1979-07-16	1	3,00	1	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
9	Ján	Hlúpy	muž	1972-04-01	2	3,00	1	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
3	Aladár	Baba	muž	1980-01-22	2	2,03	2	2	Kalifát Bagdad	Harún al-Rašíd
4	Ferdinand	Mravec	muž	1984-03-03	3	1,00	3	3	Mravenisko	Z
11	Mikuláš	Myšiak	muž	1983-06-06	5	1,66	4	4	Hollywood	Simba
12	Donald	Káčer	muž	1982-10-07	5	1,83	4	4	Hollywood	Simba
14	Peter	Pan	muž	2001-01-13	1	NULL	5	5	Neverland	NULL

Všimnime si, že vypadol jednak riadok s krajinou Haliganda, z ktorej nie je žiaden študent, a teda nebolo ju s kým skombinovať (dodržiac podmienku vo `WHERE`), jednak riadky s tými študentmi, ktorých krajina je neznáma (stĺpec `id_bydlisko` má teda hodnotu `NULL`, a preto je pravdivostná hodnota podmienky vo `WHERE` neznáma). Aby sme získali len identifikačné čísla študentov, ich mená, priezviská a názvy krajín, do časti `SELECT` klasickým spôsobom vložíme (prípadne kvalifikované) názvy príslušných stĺpcov:

```
SELECT
  študent.id,
  študent.meno,
  študent.priezvisko,
  krajina.názov
FROM
  študent,
  krajina
WHERE študent.id_bydlisko = krajina.id
```

Odpoveď:

ID	MENO	PRIEZVISKO	NÁZOV
1	Ján	Hraško	Za siedmimi horami a siedmimi dolami
2	Ružena	Šípová	Za siedmimi horami a siedmimi dolami
5	Ján	Polienko	Za siedmimi horami a siedmimi dolami
6	Juraj	Truľo	Za siedmimi horami a siedmimi dolami
9	Ján	Hlúpy	Za siedmimi horami a siedmimi dolami
3	Aladár	Baba	Kalifát Bagdad
4	Ferdinand	Mravec	Mravenisko
11	Mikuláš	Myšiak	Hollywood
12	Donald	Káčer	Hollywood
14	Peter	Pan	Neverland

Kvôli zjednodušeniu a skráteniu zápisu môžeme obom tabuľkám priradiť aliasy. (S aliasmi tabuliek sme sa už stretli v pasáži o vnorených dopytoch, zopakujme, že po premenovaní aliasom sa už nemôžeme na tabuľku odvolávať jej pôvodným menom.) Predchádzajúci dopyt teda môžeme ekvivalentne prepísať takto:

```
SELECT
  š.id,
  š.meno,
  š.priezvisko,
  k.názov
FROM
  študent AS š,
  krajina AS k
WHERE š.id_bydlisko = k.id
```

Informáciu o spojení dvoch tabuliek môžeme ekvivalentne vyjadriť aj tak, že ju vložíme do časti `FROM`:

```
SELECT
  š.id,
  š.meno,
  š.priezvisko,
  k.názov
FROM
  študent AS š
    JOIN krajina AS k ON š.id_bydlisko = k.id
```

Po kľúčovom slove `FROM` tu nenasledujú dve tabuľky, ale je tu popísaná konštrukcia jednej zložitejšej: Po názve prvej tabuľky (a jej aliase) nasleduje slovo `JOIN` („pripojiť sa“), za ním názov druhej tabuľky a po predložke `ON` („pri (splnení)“) podmienka vyjadrujúca väzbu medzi nimi. Dodajme, že v jednej klauzule `FROM` možno konštrukciu pomocou `JOIN` použiť aj viackrát, jej zrejmá asociativita (voľne (a, priznajme, nepresne) zapísaná $(x \texttt{ JOIN } y) \texttt{ JOIN } z = x \texttt{ JOIN} (y \texttt{ JOIN } z)$) dovoľuje vynechať zátvorky.

Podmienky vo `WHERE` a po `ON` môžeme kombinovať. Ak napríklad chceme študentov len z Mraveniska a Hollywoodu, popri klasickom:

```
SELECT
  š.id,
  š.meno,
  š.priezvisko,
  k.názov
FROM
  študent AS š,
  krajina AS k
WHERE
      š.id_bydlisko = k.id
  AND k.názov IN ('Mravenisko','Hollywood')
```

ekvivalentne zaberie aj dopyt:

```
SELECT
  š.id,
  š.meno,
  š.priezvisko,
  k.názov
FROM
  študent AS š
    JOIN krajina AS k ON š.id_bydlisko = k.id
WHERE k.názov IN ('Mravenisko','Hollywood')
```

Odpoveď:

ID	MENO	PRIEZVISKO	NÁZOV
4	Ferdinand	Mravec	Mravenisko
11	Mikuláš	Myšiak	Hollywood
12	Donald	Káčer	Hollywood

Vráťme sa k problému, ktorý nás priviedol k rozkladu dát do dvoch tabuliek, a to zistenie počtu študentov z jednotlivých krajín. Teraz ho už vyriešime jednoducho:

```
SELECT
  k.názov AS krajina,
  COUNT(*) AS počet_študentov
FROM
  študent AS š
    JOIN krajina AS k ON š.id_bydlisko = k.id
GROUP BY k.názov
```

Odpoveď:

KRAJINA	POČET_ŠTUDENTOV
Hollywood	2
Kalifát Bagdad	1
Mravenisko	1
Neverland	1
Za siedmimi horami a siedmimi dolami	5

Na záver slova o spojení tabuliek ešte jedna syntaktická zaujímavosť, využívaná často pri testovaní: Ak chceme vypísať všetky údaje z jednej tabuľky, do časti `SELECT` napíšeme *tabuľka*`.*`. Napríklad:

```
SELECT
  š.id,
  š.meno,
  š.priezvisko,
  k.*
FROM
  študent AS š
    JOIN krajina AS k ON š.id_bydlisko = k.id
```

Odpoveď:

ID	MENO	PRIEZVISKO	ID	NÁZOV	HLAVA
1	Ján	Hraško	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
2	Ružena	Šípová	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
5	Ján	Polienko	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
6	Juraj	Truľo	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
9	Ján	Hlúpy	1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
3	Aladár	Baba	2	Kalifát Bagdad	Harún al-Rašíd
4	Ferdinand	Mravec	3	Mravenisko	Z
11	Mikuláš	Myšiak	4	Hollywood	Simba
12	Donald	Káčer	4	Hollywood	Simba
14	Peter	Pan	5	Neverland	NULL

2.1.3 Vonkajšie spojenia

Všimnime si, že pri spojení tabuliek `študent` a `krajina` boli vylúčení tí študenti, ktorých krajina nebola známa. Ak však chceme zoznam všetkých študentov a informácia o krajine má byť len podružná, iste nám toto správanie vyhovovať nebude – neradi by sme pre chýbajúcu doplnkovú informáciu prišli o tú podstatnú. Namiesto obyčajného spojenia tabuliek tu dobre poslúži **ľavé vonkajšie spojenie**, ktoré podmienku spojenia chápe voľnejšie – pripúšťa nielen pravdivostnú hodnotu *pravda*, ale i *neznáme*. Syntakticky sa ľavé vonkajšie spojenie veľmi nelíši od obyčajného spojenia v alternatívnej forme s `JOIN`, len namiesto tohto slova napíšeme slovné spojenie `LEFT OUTER JOIN` („ľavé vonkajšie spojenie“):

```
SELECT
  š.id,
  š.meno,
  š.priezvisko,
  k.názov
FROM
  študent AS š
    LEFT OUTER JOIN krajina AS k ON š.id_bydlisko = k.id
```

Odpoveď (pridané dáta sme graficky oddelili):

ID	MENO	PRIEZVISKO	NÁZOV
1	Ján	Hraško	Za siedmimi horami a siedmimi dolami
2	Ružena	Šípová	Za siedmimi horami a siedmimi dolami
5	Ján	Polienko	Za siedmimi horami a siedmimi dolami
6	Juraj	Truľo	Za siedmimi horami a siedmimi dolami
9	Ján	Hlúpy	Za siedmimi horami a siedmimi dolami
3	Aladár	Baba	Kalifát Bagdad
4	Ferdinand	Mravec	Mravenisko
11	Mikuláš	Myšiak	Hollywood
12	Donald	Káčer	Hollywood
14	Peter	Pan	Neverland
7	Jana	Botková	NULL
8	Dana	Botková	NULL
10	Aladár	Miazga	NULL
13	Jozef	Námorník	NULL

Vidíme, že tie záznamy prvej tabuľky, ktoré nemožno napojiť na žiaden záznam druhej tabuľky, sú doplnené v stĺpcoch druhej tabuľky prázdnymi hodnotami.

Analogicky funguje aj **pravé vonkajšie spojenie**, to však doplní nespojiteľné záznamy tentoraz z druhej tabuľky prázdnymi hodnotami na mieste prvej tabuľky. Namiesto `JOIN` teraz napíšeme `RIGHT OUTER JOIN` („pravé vonkajšie spojenie“):

```
SELECT
  š.id,
  š.meno,
  š.priezvisko,
  k.názov
FROM
  študent AS š
    RIGHT OUTER JOIN krajina AS k ON š.id_bydlisko = k.id
```

Odpoveď:

ID	MENO	PRIEZVISKO	NÁZOV
1	Ján	Hraško	Za siedmimi horami a siedmimi dolami
2	Ružena	Šípová	Za siedmimi horami a siedmimi dolami
5	Ján	Polienko	Za siedmimi horami a siedmimi dolami
6	Juraj	Truľo	Za siedmimi horami a siedmimi dolami
9	Ján	Hlúpy	Za siedmimi horami a siedmimi dolami
3	Aladár	Baba	Kalifát Bagdad
4	Ferdinand	Mravec	Mravenisko
11	Mikuláš	Myšiak	Hollywood
12	Donald	Káčer	Hollywood
14	Peter	Pan	Neverland
NULL	NULL	NULL	Haliganda

Rovnaký výsledok dostaneme aj pomocou ľavého vonkajšieho spojenia, treba však vymeniť poradie tabuliek:

```
SELECT
  š.id,
  š.meno,
  š.priezvisko,
  k.názov
FROM
  krajina AS k
    LEFT OUTER JOIN študent AS š ON š.id_bydlisko = k.id
```

Posledným typom vonkajšieho prepojenia dvoch tabuliek je **úplné vonkajšie spojenie**, ktoré je zjednotením ľavého a pravého vonkajšieho spojenia. Teraz namiesto `JOIN` napíšeme `FULL OUTER JOIN` („úplné vonkajšie spojenie“):

```
SELECT
  š.id,
  š.meno,
  š.priezvisko,
  k.názov
FROM
  študent AS š
    FULL OUTER JOIN krajina AS k ON š.id_bydlisko = k.id
```

Odpoveď:

ID	MENO	PRIEZVISKO	NÁZOV
1	Ján	Hraško	Za siedmimi horami a siedmimi dolami
2	Ružena	Šípová	Za siedmimi horami a siedmimi dolami
5	Ján	Polienko	Za siedmimi horami a siedmimi dolami
6	Juraj	Truľo	Za siedmimi horami a siedmimi dolami
9	Ján	Hlúpy	Za siedmimi horami a siedmimi dolami
3	Aladár	Baba	Kalifát Bagdad
4	Ferdinand	Mravec	Mravenisko
11	Mikuláš	Myšiak	Hollywood
12	Donald	Káčer	Hollywood
14	Peter	Pan	Neverland
NULL	NULL	NULL	Haliganda
7	Jana	Botková	NULL
8	Dana	Botková	NULL
10	Aladár	Miazga	NULL
13	Jozef	Námorník	NULL

Dodajme, že (v prípade viacerých tabuliek) možno v jednom `FROM` použiť vonkajších spojení viac a môžeme ich kombinovať ako navzájom, tak s obyčajným spojením. Treba si však uvedomiť, že v takom prípade sa obvykle asociativita stráca, a preto je pri zložitejšej klauzule `FROM` istejšie používať zátvorky.

Vráťme sa ešte k prehľadu počtu študentov z jednotlivých krajín. Ak v ňom chceme mať aj tie krajiny, z ktorých nepochádza nik, použijeme vonkajšie spojenie, tentoraz pravé:

```
SELECT
  k.názov AS krajina,
  COUNT(š.priezvisko) AS počet_študentov
FROM
  študent AS š
    RIGHT OUTER JOIN krajina AS k ON š.id_bydlisko = k.id
GROUP BY k.názov
```

Odpoveď:

KRAJINA	POČET_ŠTUDENTOV
Haliganda	0
Hollywood	2
Kalifát Bagdad	1
Mravenisko	1
Neverland	1
Za siedmimi horami a siedmimi dolami	5

Treba zdôrazniť, že namiesto `COUNT(š.priezvisko)` nemožno uviesť `COUNT(*)`, do počtu študentov z Haligandy by sa totiž nesprávne započítala prázdna hodnota:

```
SELECT
  k.názov AS krajina,
  COUNT(*) AS klamný_počet_študentov
FROM
  študent AS š
    RIGHT OUTER JOIN krajina AS k ON š.id_bydlisko = k.id
GROUP BY k.názov
```

Odpoveď:

KRAJINA	KLAMNÝ_POČET_ŠTUDENTOV
Haliganda	1
Hollywood	2
Kalifát Bagdad	1
Mravenisko	1
Neverland	1
Za siedmimi horami a siedmimi dolami	5

A ešte jeden zádrheľ vonkajšieho spojenia: Ak chceme počty študentov len z krajín začínajúcich sa na `H`, jednoducho doplníme príslušnú podmienku:

```
SELECT
  k.názov AS krajina,
  COUNT(š.priezvisko) AS počet_študentov
FROM
  študent AS š
    RIGHT OUTER JOIN krajina AS k ON š.id_bydlisko = k.id
WHERE k.názov LIKE 'H%'
GROUP BY k.názov
```

Odpoveď:

KRAJINA	POČET_ŠTUDENTOV
Haliganda	0
Hollywood	2

Pri obyčajnom spojení by sme mohli doplnenú podmienku ekvivalentne presunúť k `JOIN`, tu však analogický dopyt:

```
SELECT
  k.názov AS krajina,
  COUNT(š.priezvisko) AS počet_študentov
FROM
  študent AS š
    RIGHT OUTER JOIN krajina AS k ON š.id_bydlisko = k.id AND k.názov LIKE 'H%'
GROUP BY k.názov
```

dáva odlišnú odpoveď:

KRAJINA	POČET_ŠTUDENTOV
Haliganda	0
Hollywood	2
Kalifát Bagdad	0
Mravenisko	0
Neverand	0
Za siedmimi horami a siedmimi dolami	0

Pri používaní vonkajších spojení preto treba zvýšenú opatrnosť.

Na záver dodajme, že na zdôraznenie odlišnosti obyčajného spojenia od vonkajších mu hovoríme aj **vnútorné**. Môžeme to dokonca vyjadriť aj v samotnom dopyte, ak pred `JOIN` doplníme slovíčko `INNER` („vnútorné“). Nasledujúce dva dopyty sú teda ekvivalentné:

```
SELECT
  š.id,
  š.meno,
  š.priezvisko,
  k.názov
FROM
  študent AS š
    JOIN krajina AS k ON k.id = š.id_bydlisko
```

a:

```
SELECT
  š.id,
  š.meno,
  š.priezvisko,
  k.názov
FROM
  študent AS š
    INNER JOIN krajina AS k ON k.id = š.id_bydlisko
```

2.2 Cudzí kľúč

2.2.1 Načo je cudzí kľúč?

Predstavme si, že niekto z tabuľky krajina (či už nedopatrením, alebo zámerne) vymaže záznam krajiny Za siedmimi horami a siedmimi dolami. V tom okamihu sa údaje stanú (v našom chápaní) nekonzistentné, lebo stĺpec id_bydlisko z tabuľky študent v záznamoch študentov Hraška, Šípovej, Polienka, Truľa a Hlúpeho z tejto krajiny už nemôžeme chápať ako ukazovateľ na príslušné id z tabuľky krajina, ukazovali by totiž do prázdna (uvedomme si, že je to niečo iné než pri záznamoch, ktoré majú v stĺpci id_bydlisko prázdnu hodnotu, tie nemajú takú ambíciu). Databáza je v tom nevinne – veď o tom, že stĺpce študent.id_bydlisko a krajina.id by mali nejako súvisieť, sme ju neinformovali.

Ako teda prenechať starostlivosť o súvis týchto dvoch stĺpcov databázovému systému? Na scénu tak vstupuje nové integritné obmedzenie zvané **cudzí kľúč**. Pri definícii jednej tabuľky musíme príslušnému stĺpcu jemne naznačiť, že všetky jeho hodnoty sa musia vyskytovať v zodpovedajúcom stĺpci druhej tabuľky (alebo byť prázdne (pravdaže, len vtedy, ak definícia stĺpca hodnoty NULL pripúšťa)). Z toho by mala vyplývať zhodnosť ich dátových typov, no pripúšťajú sa i výnimky (napríklad ukazovanie DEC na INT a naopak), ktorým je však najistejšie sa vyhýbať. Pod cudzím kľúčom potom rozumieme vzťah medzi týmito dvoma stĺpcami, resp. (zjednodušene) ľubovoľný z nich. Samozrejme, definovanie cudzieho kľúča predpokladá, že tabuľka, na ktorú má odkazovať, už existuje (alebo sa práve vytvára a ukazuje sama na seba). Uvedomme si, že **odkazovaný stĺpec musí byť jednoznačný** (a teda musí byť kľúčom, či už primárnym, alebo aspoň sekundárnym), inak by vznikla nejednoznačnosť, na ktorý z riadkov má potenciálny odkaz ukazovať.

Syntakticky môžeme cudzí kľúč vyjadriť viacerými spôsobmi. Jedna z možností je, že na koniec definície príslušného stĺpca prvej tabuľky napíšeme slovo REFERENCES („odkazuje“) nasledované názvom druhej tabuľky a v zátvorkách menom príslušného jej stĺpca (ktorý, ako sme už povedali, musí byť PRIMARY KEY, alebo aspoň UNIQUE). V našom prípade hrá úlohu prvého stĺpca študent.id_bydlisko a druhým je stĺpec krajina.id. Takže:

```
DROP TABLE študent
;
CREATE TABLE študent
(
  id              INT         NOT NULL PRIMARY KEY,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4)     NOT NULL,
  dátum_narodenia DATE        NOT NULL,
  ročník          INT         NOT NULL,
  priemer         DEC(3,2),
  id_bydlisko     INT                 REFERENCES krajina (id)
)
```

Spravidla je stĺpec druhej tabuľky jej primárnym kľúčom, v takom prípade môžeme zátvorky s menom stĺpca vynechať. Tak je to aj v prípade odkazovaného stĺpca krajina.id, alternatívnym zápisom je teda:

```
DROP TABLE študent
;
CREATE TABLE študent
(
  id              INT         NOT NULL PRIMARY KEY,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4)     NOT NULL,
  dátum_narodenia DATE        NOT NULL,
  ročník          INT         NOT NULL,
  priemer         DEC(3,2),
  id_bydlisko     INT                 REFERENCES krajina
)
```

A ešte jeden ekvivalentný spôsob, keď toto integritné obmedzenie uvedieme až po definícii všetkých stĺpcov, a to slovným spojením FOREIGN KEY („cudzí kľúč“), za ktorým nasleduje meno odkazujúceho stĺpca a po ňom už spomínané REFERENCES s menom odkazovanej tabuľky a v zátvorkách odkazovaného stĺpca. Ak to je primárny kľúč, aj tu ho možno vynechať. V našom prípade teda:

```
DROP TABLE študent
;
CREATE TABLE študent
(
  id              INT         NOT NULL PRIMARY KEY,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4)     NOT NULL,
  dátum_narodenia DATE        NOT NULL,
  ročník          INT         NOT NULL,
  priemer         DEC(3,2),
  id_bydlisko     INT,
  FOREIGN KEY (id_bydlisko) REFERENCES krajina (id)
)
```

resp.:

```
DROP TABLE študent
;
CREATE TABLE študent
(
  id              INT         NOT NULL PRIMARY KEY,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4)     NOT NULL,
  dátum_narodenia DATE        NOT NULL,
  ročník          INT         NOT NULL,
  priemer         DEC(3,2),
  id_bydlisko     INT,
  FOREIGN KEY (id_bydlisko) REFERENCES krajina
)
```

Cudzí kľúč môže pozostávať aj z viacerých stĺpcov, ktoré potom odkazujú na rovnaký počet stĺpcov rovnakého dátového typu v príslušnom poradí. V takom prípade môžeme použiť iba tento posledný spôsob vyjadrenia cudzieho kľúča. V zátvorkách po slovnom spojení FOREIGN KEY sú potom odkazujúce stĺpce oddelené čiarkami a v zátvorkách po mene odkazovanej tabuľky odkazované stĺpce v príslušnom poradí, tiež oddelené čiarkami. Ak odkazované stĺpce tvoria zároveň primárny kľúč odkazovanej tabuľky, túto druhú zátvorku možno celú vynechať. Hodnoty takéhoto viacnásobného cudzieho kľúča v odkazujúcej tabuľke sa potom musia vyskytovať (ako celok) v odkazujúcej tabuľke, ale vyhovujúca je aj možnosť, že aspoň jedna z nich je prázdna.

Dodajme ešte, že na rozdiel od primárneho kľúča **môže byť v tabuľke cudzích kľúčov viac**. Jeden stĺpec môže byť súčasťou viacerých cudzích kľúčov, môže dokonca ukazovať aj na viacero rôznych stĺpcov, a to aj v tej istej tabuľke (ba dokonca teoreticky aj sám na seba, ale to je zmysluprázdne).

Vyčistime stôl:

```
DROP TABLE študent
;
DROP TABLE krajina
```

Ak sa teraz pokúsime vytvoriť tabuľku študent, zlyháme, lebo odkazovaná tabuľka krajina ešte neexistuje. Pri tabuľkách zviazaných cudzím kľúčom teda musíme dávať pozor aj na poradie ich vytvárania – najprv odkazovaná, potom odkazujúca. V našom prípade teda najprv krajina, a až potom študent:

```
CREATE TABLE krajina
(
  id              INT         NOT NULL PRIMARY KEY,
```

```
  názov           VARCHAR(50) NOT NULL,
  hlava           VARCHAR(50)
)
;
CREATE TABLE študent
(
  id              INT         NOT NULL PRIMARY KEY,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4)     NOT NULL,
  dátum_narodenia DATE        NOT NULL,
  ročník          INT         NOT NULL,
  priemer         DEC(3,2),
  id_bydlisko     INT                 REFERENCES krajina
)
```

2.2.2 Správanie cudzieho kľúča pri vkladaní dát

Skúsme do tabuliek vložiť údaje. Opäť však nemôžeme začať tabuľkou `študent`, lebo (neprázdne) údaje v stĺpci `id_bydlisko` by sa pokúšali ukazovať na neexistujúce záznamy v tabuľke `krajina` (tá je prázdna), čo cudzí kľúč nedovolí. Východisko je jednoznačné – najprv treba naplniť tabuľku `krajina`, a až potom tabuľku `študent`:

```
INSERT INTO krajina
VALUES
  (1, 'Za siedmimi horami a siedmimi dolami', 'Drozdia Brada' ),
  (2, 'Kalifát Bagdad',                       'Harún al-Rašíd'),
  (3, 'Mravenisko',                           'Z'             ),
  (4, 'Hollywood',                            'Simba'         ),
  (5, 'Neverland',                            NULL            ),
  (6, 'Haliganda',                            NULL            )
;
INSERT INTO študent
VALUES
  (1,  'Ján',       'Hraško',   'muž',  '12.7.1987',  1, 1.83, 1   ),
  (2,  'Ružena',    'Šípová',   'žena', '1.2.1984',   1, 1.22, 1   ),
  (3,  'Aladár',    'Baba',     'muž',  '22.1.1980',  2, 2.03, 2   ),
  (4,  'Ferdinand', 'Mravec',   'muž',  '3.3.1984',   3, 1.00, 3   ),
  (5,  'Ján',       'Polienko', 'muž',  '14.4.1982',  5, 2.28, 1   ),
  (6,  'Juraj',     'Truľo',    'muž',  '16.7.1979',  1, 3.00, 1   ),
  (7,  'Jana',      'Botková',  'žena', '21.9.1977',  4, 1.50, NULL),
  (8,  'Dana',      'Botková',  'žena', '21.9.1977',  4, 1.40, NULL),
  (9,  'Ján',       'Hlúpy',    'muž',  '1.4.1972',   2, 3.00, 1   ),
  (10, 'Aladár',    'Miazga',   'muž',  '22.12.1987', 3, 2.06, NULL),
  (11, 'Mikuláš',   'Myšiak',   'muž',  '6.6.1983',   5, 1.66, 4   ),
  (12, 'Donald',    'Káčer',    'muž',  '7.10.1982',  5, 1.83, 4   ),
  (13, 'Jozef',     'Námorník', 'muž',  '23.9.1981',  2, 2.90, NULL),
  (14, 'Peter',     'Pan',      'muž',  '13.1.2001',  1, NULL, 5   )
```

Keďže každá z (neprázdnych) hodnôt stĺpca `študent.id_bydlisko` sa už v čase vkladania vyskytuje v stĺpci `krajina.id`, cudzí kľúč túto operáciu schváli.

Ukázali sme si, že pri tabuľkách zviazaných cudzím kľúčom musíme dávať pozor aj na poradie vkladania záznamov a dodržať pravidlo, že cudzí kľúč nesmie ukazovať „do prázdna“. To však neznamená, že do tabuľky `krajina` už po vložení záznamov do tabuľky `študent` nemôžeme opäť niečo vložiť. Takýto príkaz teda prejde bez problémov, veď uvedené pravidlo sa tým neporuší:

```
INSERT INTO krajina
VALUES (7, 'Půlnoční království', 'Miroslav')
```

2.2.3 Správanie cudzieho kľúča pri modifikácii dát

Vyskúšajme teraz reakciu cudzieho kľúča na modifikáciu dát. Zmeniť hodnotu odkazujúceho stĺpca na inú hodnotu, ktorá sa v odkazovanom stĺpci vyskytuje, je možné – napr. príkaz, ktorý vyjadruje, že Janko Hraško sa presťahoval do krajiny Půlnoční království:

```
UPDATE študent
SET id_bydlisko = 7
WHERE id = 1
```

prejde. Ak by sme sa však pokúsili hodnotu odkazujúceho stĺpca zmeniť na hodnotu, ktorá sa v odkazovanom stĺpci nevyskytuje (pravdaže, s výnimkou prázdnej hodnoty), príkaz zlyhá:

```
UPDATE študent
SET id_bydlisko = 8
WHERE id = 1
```

Podobne zlyhá pokus o zmenu odkazovanej hodnoty stĺpca `id` v tabuľke `krajina`:

```
UPDATE krajina
SET id = 8
WHERE id = 1
```

pretože po odstránení identifikátora `1` by niektorí študenti bez vedomia stratili svoje bydlisko. Zato podobný príkaz:

```
UPDATE krajina
SET id = 8
WHERE id = 6
```

by nezlyhal, lebo na Haligandu s číslom `6` neukazuje žiaden záznam z tabuľky `študent`.

2.2.4 Tri typy správania cudzieho kľúča pri mazaní dát

Cudzí kľúč je v pohotovosti aj pri odstraňovaní záznamov. Mazanie z tabuľky `študent` nemá byť prečo problematické, odkazovaný nie je žiaden jej záznam. Inak je to s tabuľkou `krajina`. Keďže nik zo študentov nepochádza z Haligandy, jej záznam (už s pred chvíľkou zmenenou hodnotou `id`) sa nám podarí z tabuľky `krajina` vymazať bez porušenia spomenutého pravidla:

```
DELETE FROM krajina
WHERE id = 8
```

Horšie to však bude s ostatnými záznamami – na každý z nich totiž niečo ukazuje. Keďže odstránením ľubovoľného z nich by sa pravidlo o cudzom kľúči porušilo, nemôžeme žiaden z nich vymazať (prv než vymažeme všetky na neho ukazujúce záznamy). Preto napr. takýto príkaz zlyhá:

```
DELETE FROM krajina
WHERE id = 1
```

Ak však predtým z tabuľky `študent` vymažeme všetky záznamy, ktoré naň ukazujú:

```
DELETE FROM študent
WHERE id_bydlisko = 1
```

predchádzajúci príkaz prejde – na mazaný záznam už totiž nič neukazuje.

Toto je však len jeden z možných módov práce cudzieho kľúča pri mazaní – ten najopatrnejší. Prípustné je však aj rozšafnejšie správanie – vymazanie záznamu v odkazovanej tabuľke je povolené, ak spôsobí aj vymazanie všetkých záznamov, ktoré naň ukazujú (ak tomu, pravdaže, nebráni iné integritné obmedzenie). V takom prípade však musíme toto správanie deklarovať už pri definícii tabuľky, a to tak, že na konci definície cudzieho kľúča dodáme slovné spojenie `ON DELETE CASCADE` („pri mazaní (sa správaj) kaskádovo“):

```
DROP TABLE študent
;
DROP TABLE krajina
;
CREATE TABLE krajina
(
  id              INT         NOT NULL PRIMARY KEY,
  názov           VARCHAR(50) NOT NULL,
  hlava           VARCHAR(50)
)
;
CREATE TABLE študent
(
  id              INT         NOT NULL PRIMARY KEY,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4)     NOT NULL,
  dátum_narodenia DATE        NOT NULL,
  ročník          INT         NOT NULL,
  priemer         DEC(3,2),
  id_bydlisko     INT                 REFERENCES krajina ON DELETE CASCADE
)
;
INSERT INTO krajina
VALUES
  (1, 'Za siedmimi horami a siedmimi dolami', 'Drozdia Brada' ),
  (2, 'Kalifát Bagdad',                       'Harún al-Rašíd'),
  (3, 'Mravenisko',                           'Z'             ),
  (4, 'Hollywood',                            'Simba'         ),
  (5, 'Neverland',                            NULL            ),
  (6, 'Haliganda',                            NULL            )
;
INSERT INTO študent
VALUES
  (1,  'Ján',       'Hraško',   'muž',  '12.7.1987',  1, 1.83, 1   ),
  (2,  'Ružena',    'Šípová',   'žena', '1.2.1984',   1, 1.22, 1   ),
  (3,  'Aladár',    'Baba',     'muž',  '22.1.1980',  2, 2.03, 2   ),
  (4,  'Ferdinand', 'Mravec',   'muž',  '3.3.1984',   3, 1.00, 3   ),
  (5,  'Ján',       'Polienko', 'muž',  '14.4.1982',  5, 2.28, 1   ),
  (6,  'Juraj',     'Truľo',    'muž',  '16.7.1979',  1, 3.00, 1   ),
  (7,  'Jana',      'Botková',  'žena', '21.9.1977',  4, 1.50, NULL),
  (8,  'Dana',      'Botková',  'žena', '21.9.1977',  4, 1.40, NULL),
  (9,  'Ján',       'Hlúpy',    'muž',  '1.4.1972',   2, 3.00, 1   ),
  (10, 'Aladár',    'Miazga',   'muž',  '22.12.1987', 3, 2.06, NULL),
  (11, 'Mikuláš',   'Myšiak',   'muž',  '6.6.1983',   5, 1.66, 4   ),
  (12, 'Donald',    'Káčer',    'muž',  '7.10.1982',  5, 1.83, 4   ),
  (13, 'Jozef',     'Námorník', 'muž',  '23.9.1981',  2, 2.90, NULL),
  (14, 'Peter',     'Pan',      'muž',  '13.1.2001',  1, NULL, 5   )
```

Vyskúšajme teraz príkaz, ktorý by v predchádzajúcom prípade zlyhal, pretože na mazaný záznam ukazuje aspoň jeden ďalší záznam:

```
DELETE FROM krajina
WHERE id = 1
```

Ako teraz vyzerá tabuľka `krajina`, nás veľmi neprekvapí:

```
SELECT *
FROM krajina
```

Odpoveď:

ID	NÁZOV	HLAVA
2	Kalifát Bagdad	Harún al-Rašíd
3	Mravenisko	Z
4	Hollywood	Simba
5	Neverland	NULL
6	Haliganda	NULL

Tabuľka `študent` je zaujímavejšia, vypadli z nej totiž všetky záznamy, ktoré mali v stĺpci `id_bydlisko` mazanú hodnotu `1`:

```
SELECT *
FROM študent
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER	ID_BYDLISKO
3	Aladár	Baba	muž	1980-01-22	2	2,03	2
4	Ferdinand	Mravec	muž	1984-03-03	3	1,00	3
7	Jana	Botková	žena	1977-09-21	4	1,50	NULL
8	Dana	Botková	žena	1977-09-21	4	1,40	NULL
10	Aladár	Miazga	muž	1987-12-22	3	2,06	NULL
11	Mikuláš	Myšiak	muž	1983-06-06	5	1,66	4
12	Donald	Káčer	muž	1982-10-07	5	1,83	4
13	Jozef	Námorník	muž	1981-09-23	2	2,90	NULL
14	Peter	Pan	muž	2001-01-13	1	NULL	5

Pomenovanie **kaskádové mazanie** je namieste. Ak je totiž takýmto spôsobom definovaný aj nejaký iný cudzí kľúč v odkazujúcej tabuľke (a ona je teda odkazovanou pre nejakú tretiu tabuľku), vymazanie jej záznamov môže spôsobiť vymazanie ešte ďalších záznamov (z tej tretej tabuľky), ktoré na ne ukazujú, atď.. Jedno nevinné vymazanie jediného záznamu tak môže spôsobiť v celej databáze dominový efekt, ktorý môže (ale nemusí) byť nežiaduci. Neznamená to však, že toto správanie je nesprávne, no pri použití tohto spôsobu treba byť nadmieru opatrný. Aj tu však platí pravidlo „take it or leave it“ – buď sa príkaz kaskádového mazania vykoná celý, alebo sa nevykoná vôbec: **Ak nie je (z ľubovoľných dôvodov) dovolená ktorákoľvek časť transakcie, nevykoná sa ani žiadna jej sama osebe prípustná časť.**

Tretí, kompromisný typ správania cudzieho kľúča pri mazaní je taký, že odkazujúce záznamy nebudú vymazané, ale hodnoty odkazujúceho stĺpca v nich sa zmenia na prázdnu hodnotu. Pri definícii cudzieho kľúča to vyjadríme tak, že na jeho koniec tentoraz uvedieme slovné spojenie `ON DELETE SET NULL` („pri mazaní nastav prázdnu hodnotu“). Nutnou podmienkou však je, aby dotyčný stĺpec pripúšťal prázdne hodnoty, inak príkaz na zostrojenie tabuľky zlyhá:

```
DROP TABLE študent
;
DROP TABLE krajina
;
CREATE TABLE krajina
(
  id              INT         NOT NULL PRIMARY KEY,
  názov           VARCHAR(50) NOT NULL,
  hlava           VARCHAR(50)
)
;
CREATE TABLE študent
(
  id              INT         NOT NULL PRIMARY KEY,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4)     NOT NULL,
  dátum_narodenia DATE        NOT NULL,
  ročník          INT         NOT NULL,
  priemer         DEC(3,2),
  id_bydlisko     INT                 REFERENCES krajina ON DELETE SET NULL
)
```

Vložíme dáta:

```
INSERT INTO krajina
VALUES
  (1, 'Za siedmimi horami a siedmimi dolami', 'Drozdia Brada' ),
  (2, 'Kalifát Bagdad',                       'Harún al-Rašíd'),
  (3, 'Mravenisko',                           'Z'             ),
  (4, 'Hollywood',                            'Simba'         ),
  (5, 'Neverland',                            NULL            ),
  (6, 'Haliganda',                            NULL            )
;
INSERT INTO študent
VALUES
  (1,  'Ján',       'Hraško',   'muž',  '12.7.1987',  1, 1.83, 1   ),
  (2,  'Ružena',    'Šípová',   'žena', '1.2.1984',   1, 1.22, 1   ),
  (3,  'Aladár',    'Baba',     'muž',  '22.1.1980',  2, 2.03, 2   ),
  (4,  'Ferdinand', 'Mravec',   'muž',  '3.3.1984',   3, 1.00, 3   ),
  (5,  'Ján',       'Polienko', 'muž',  '14.4.1982',  5, 2.28, 1   ),
  (6,  'Juraj',     'Truľo',    'muž',  '16.7.1979',  1, 3.00, 1   ),
  (7,  'Jana',      'Botková',  'žena', '21.9.1977',  4, 1.50, NULL),
  (8,  'Dana',      'Botková',  'žena', '21.9.1977',  4, 1.40, NULL),
  (9,  'Ján',       'Hlúpy',    'muž',  '1.4.1972',   2, 3.00, 1   ),
  (10, 'Aladár',    'Miazga',   'muž',  '22.12.1987', 3, 2.06, NULL),
  (11, 'Mikuláš',   'Myšiak',   'muž',  '6.6.1983',   5, 1.66, 4   ),
  (12, 'Donald',    'Káčer',    'muž',  '7.10.1982',  5, 1.83, 4   ),
  (13, 'Jozef',     'Námorník', 'muž',  '23.9.1981',  2, 2.90, NULL),
  (14, 'Peter',     'Pan',      'muž',  '13.1.2001',  1, NULL, 5   )
```

Teraz príkaz mazania:

```
DELETE FROM krajina
WHERE id = 1
```

Tabuľka `krajina` sa zmenila rovnako ako minule:

```
SELECT *
FROM krajina
```

Odpoveď:

ID	NÁZOV	HLAVA
2	Kalifát Bagdad	Harún al-Rašíd
3	Mravenisko	Z
4	Hollywood	Simba
5	Neverland	NULL
6	Haliganda	NULL

V tabuľke `študent` však nastali iné zmeny – hodnoty stĺpca `id_bydlisko` v zasiahnutých riadkoch majú hodnotu `NULL`:

```
SELECT *
FROM študent
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER	ID_BYDLISKO
1	Ján	Hraško	muž	1987-07-12	1	1,83	NULL
2	Ružena	Šípová	žena	1984-02-01	1	1,22	NULL
3	Aladár	Baba	muž	1980-01-22	2	2,03	2
4	Ferdinand	Mravec	muž	1984-03-03	3	1,00	3
5	Ján	Polienko	muž	1982-04-14	5	2,28	NULL
6	Juraj	Truľo	muž	1979-07-16	1	3,00	NULL
7	Jana	Botková	žena	1977-09-21	4	1,50	NULL
8	Dana	Botková	žena	1977-09-21	4	1,40	NULL
9	Ján	Hlúpy	muž	1972-04-01	2	3,00	NULL
10	Aladár	Miazga	muž	1987-12-22	3	2,06	NULL
11	Mikuláš	Myšiak	muž	1983-06-06	5	1,66	4
12	Donald	Káčer	muž	1982-10-07	5	1,83	4
13	Jozef	Námorník	muž	1981-09-23	2	2,90	NULL
14	Peter	Pan	muž	2001-01-13	1	NULL	5

2.2.5 Správanie cudzieho kľúča pri mazaní odkazovanej tabuľky

Na záver podkapitoly straťme slovo o mazaní tabuliek zviazaných cudzím kľúčom. Odkazovanú tabuľku (teda nie len jej odkaz) možno vymazať vždy, zároveň s ňou však budú vymazané aj cudzie kľúče v iných tabuľkách, ktoré na ňu odkazujú. Ak by sme v našom príklade teda vymazali tabuľku `krajina`:

```
DROP TABLE krajina
```

v tabuľke `študent` už nemusíme brať cudzí kľúč do úvahy, a teda hodnoty jej stĺpca `id_bydlisko` môžu byť ľubovoľné – veď stratili význam. Napr. nezlyhá príkaz:

```
INSERT INTO študent
VALUES (15, 'Sleepy', 'Dwarf', 'muž', '2.12.1292', 1, NULL, 100)
```

2.3 Modifikácia štruktúry tabuľky

Neraz sa už ukázalo, že štruktúra navrhovanej tabuľky prestala byť vo svetle nových faktov či požiadaviek vyhovujúca. Riešili sme to jednoducho – tabuľku sme prosto vymazali a vytvorili nanovo. Už vtedy sme však tušili (a naznačovali), že toto riešenie nemusí byť vždy najšťastnejšie – na istú chvíľu strácame všetky záznamy. Ukážeme si preto také spôsoby zmeny štruktúry, pri ktorých sa údaje netratia, budeme sa teda zaoberať rôznymi verziami príkazu `ALTER TABLE` („zmeň tabuľku“).

2.3.1 Pridávanie a modifikácia stĺpcov

Predpokladajme, že naša tabuľka `študent` má takúto štruktúru:

```
DROP TABLE študent
;
CREATE TABLE študent
(
  id              INT         NOT NULL PRIMARY KEY,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4)     NOT NULL,
  dátum_narodenia DATE        NOT NULL,
  ročník          INT         NOT NULL,
  priemer         DEC(3,2),
  id_bydlisko     INT
)
```

a takéto dáta:

```
INSERT INTO študent
VALUES
  (1,  'Ján',       'Hraško',   'muž',  '12.7.1987',  1, 1.83, 1   ),
  (2,  'Ružena',    'Šípová',   'žena', '1.2.1984',   1, 1.22, 1   ),
  (3,  'Aladár',    'Baba',     'muž',  '22.1.1980',  2, 2.03, 2   ),
  (4,  'Ferdinand', 'Mravec',   'muž',  '3.3.1984',   3, 1.00, 3   ),
  (5,  'Ján',       'Polienko', 'muž',  '14.4.1982',  5, 2.28, 1   ),
  (6,  'Juraj',     'Truľo',    'muž',  '16.7.1979',  1, 3.00, 1   ),
  (7,  'Jana',      'Botková',  'žena', '21.9.1977',  4, 1.50, NULL),
  (8,  'Dana',      'Botková',  'žena', '21.9.1977',  4, 1.40, NULL),
  (9,  'Ján',       'Hlúpy',    'muž',  '1.4.1972',   2, 3.00, 1   ),
  (10, 'Aladár',    'Miazga',   'muž',  '22.12.1987', 3, 2.06, NULL),
  (11, 'Mikuláš',   'Myšiak',   'muž',  '6.6.1983',   5, 1.66, 4   ),
  (12, 'Donald',    'Káčer',    'muž',  '7.10.1982',  5, 1.83, 4   ),
  (13, 'Jozef',     'Námorník', 'muž',  '23.9.1981',  2, 2.90, NULL),
  (14, 'Peter',     'Pan',      'muž',  '13.1.2001',  1, NULL, 5   )
```

Ak by sme chceli o študentoch evidovať aj to, či sú fajčiari, je vhodné do tabuľky pridať rovnomenný stĺpec. Keďže jeho hodnoty môžu byť len `áno` alebo `nie`, vhodný dátový typ preň bude `CHAR` s rozsahom `3`. Príslušný príkaz vyzerá takto:

```
ALTER TABLE študent ADD COLUMN fajčiar CHAR(3)
```

alebo alternatívne:

```
ALTER TABLE študent ADD fajčiar CHAR(3)
```

Za menom tabuľky teda nasleduje slovné spojenie `ADD COLUMN` („pridaj stĺpec“) alebo prosté `ADD` („pridaj“), za ktorým sa nachádza definícia stĺpca, tak ako sme na ňu zvyknutí. Obsah tabuľky je teraz takýto:

```
SELECT *
FROM študent
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER	ID_BYDLISKO	FAJČIAR
1	Ján	Hraško	muž	1987-07-12	1	1,83	1	NULL
2	Ružena	Šípová	žena	1984-02-01	1	1,22	1	NULL
3	Aladár	Baba	muž	1980-01-22	2	2,03	2	NULL
4	Ferdinand	Mravec	muž	1984-03-03	3	1,00	3	NULL
5	Ján	Polienko	muž	1982-04-14	5	2,28	1	NULL
6	Juraj	Truľo	muž	1979-07-16	1	3,00	1	NULL
7	Jana	Botková	žena	1977-09-21	4	1,50	NULL	NULL
8	Dana	Botková	žena	1977-09-21	4	1,40	NULL	NULL
9	Ján	Hlúpy	muž	1972-04-01	2	3,00	1	NULL
10	Aladár	Miazga	muž	1987-12-22	3	2,06	NULL	NULL
11	Mikuláš	Myšiak	muž	1983-06-06	5	1,66	4	NULL
12	Donald	Káčer	muž	1982-10-07	5	1,83	4	NULL
13	Jozef	Námorník	muž	1981-09-23	2	2,90	NULL	NULL
14	Peter	Pan	muž	2001-01-13	1	NULL	5	NULL

Naozaj teda pribudol nový stĺpec `fajčiar` a každý záznam má v ňom prázdnu hodnotu. Definícia pridávaného stĺpca však môže byť aj komplikovanejšia, môže zahŕňať ľubovoľné integritné obmedzenie (dokonca aj primárny kľúč, ak ho pôvodná tabuľka nemala). Ak uvážime, že `fajčiar` musí mať nejakú hodnotu, predchádzajúce pridanie stĺpca radšej upravíme. Najprv však návrat do pôvodného stavu:

```
DROP TABLE študent
;
CREATE TABLE študent
(
  id              INT         NOT NULL PRIMARY KEY,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4)     NOT NULL,
  dátum_narodenia DATE        NOT NULL,
  ročník          INT         NOT NULL,
  priemer         DEC(3,2),
  id_bydlisko     INT
)
;
INSERT INTO študent
VALUES
  (1,  'Ján',       'Hraško',   'muž',  '12.7.1987',  1, 1.83, 1   ),
  (2,  'Ružena',    'Šípová',   'žena', '1.2.1984',   1, 1.22, 1   ),
  (3,  'Aladár',    'Baba',     'muž',  '22.1.1980',  2, 2.03, 2   ),
  (4,  'Ferdinand', 'Mravec',   'muž',  '3.3.1984',   3, 1.00, 3   ),
  (5,  'Ján',       'Polienko', 'muž',  '14.4.1982',  5, 2.28, 1   ),
  (6,  'Juraj',     'Truľo',    'muž',  '16.7.1979',  1, 3.00, 1   ),
  (7,  'Jana',      'Botková',  'žena', '21.9.1977',  4, 1.50, NULL),
  (8,  'Dana',      'Botková',  'žena', '21.9.1977',  4, 1.40, NULL),
  (9,  'Ján',       'Hlúpy',    'muž',  '1.4.1972',   2, 3.00, 1   ),
  (10, 'Aladár',    'Miazga',   'muž',  '22.12.1987', 3, 2.06, NULL),
  (11, 'Mikuláš',   'Myšiak',   'muž',  '6.6.1983',   5, 1.66, 4   ),
  (12, 'Donald',    'Káčer',    'muž',  '7.10.1982',  5, 1.83, 4   ),
  (13, 'Jozef',     'Námorník', 'muž',  '23.9.1981',  2, 2.90, NULL),
  (14, 'Peter',     'Pan',      'muž',  '13.1.2001',  1, NULL, 5   )
```

Nasledujúci príkaz zlyhá, lebo stĺpcu `fajčiar` nepriamo nanucujeme prázdne hodnoty, ktoré nemôže prijať:

```
ALTER TABLE študent ADD fajčiar CHAR(3) NOT NULL
```

Vyriešime to však jednoducho – doplníme preddefinovanú hodnotu. Vyberieme si `nie`, veď každý sa rodí ako nefajčiar. . . :

```
ALTER TABLE študent ADD fajčiar CHAR(3) NOT NULL DEFAULT 'nie'
```

Tabuľka teraz vyzerá takto:

```
SELECT *
FROM študent
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER	ID_BYDLISKO	FAJČIAR
1	Ján	Hraško	muž	1987-07-12	1	1,83	1	nie
2	Ružena	Šípová	žena	1984-02-01	1	1,22	1	nie
3	Aladár	Baba	muž	1980-01-22	2	2,03	2	nie
4	Ferdinand	Mravec	muž	1984-03-03	3	1,00	3	nie
5	Ján	Polienko	muž	1982-04-14	5	2,28	1	nie
6	Juraj	Truľo	muž	1979-07-16	1	3,00	1	nie
7	Jana	Botková	žena	1977-09-21	4	1,50	NULL	nie
8	Dana	Botková	žena	1977-09-21	4	1,40	NULL	nie
9	Ján	Hlúpy	muž	1972-04-01	2	3,00	1	nie
10	Aladár	Miazga	muž	1987-12-22	3	2,06	NULL	nie
11	Mikuláš	Myšiak	muž	1983-06-06	5	1,66	4	nie
12	Donald	Káčer	muž	1982-10-07	5	1,83	4	nie
13	Jozef	Námorník	muž	1981-09-23	2	2,90	NULL	nie
14	Peter	Pan	muž	2001-01-13	1	NULL	5	nie

Isteže, dáta teraz nezodpovedajú skutočnosti (veď napríklad Pepek Námorník je bez svojej fajky nepredstaviteľný), ale nič nám nebráni ich modifikovať. Pretože máme v princípe iba dve možnosti (i keď niektorí fajčiari svoju bizarnú úchylku zľahčujú prívlastkom „príležitostný“), radšej sme mali predchádzajúci príkaz doplniť príslušnou kontrolou:

```
ALTER TABLE študent ADD fajčiar CHAR(3) NOT NULL DEFAULT 'nie' CHECK (fajčiar IN ('áno','nie'))
```

Vráťme sa do pôvodného stavu:

```
DROP TABLE študent
;
CREATE TABLE študent
(
  id              INT         NOT NULL PRIMARY KEY,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4)     NOT NULL,
  dátum_narodenia DATE        NOT NULL,
  ročník          INT         NOT NULL,
  priemer         DEC(3,2),
  id_bydlisko     INT
)
;
INSERT INTO študent
VALUES
  (1,  'Ján',       'Hraško',   'muž',  '12.7.1987',  1, 1.83, 1   ),
  (2,  'Ružena',    'Šípová',   'žena', '1.2.1984',   1, 1.22, 1   ),
  (3,  'Aladár',    'Baba',     'muž',  '22.1.1980',  2, 2.03, 2   ),
  (4,  'Ferdinand', 'Mravec',   'muž',  '3.3.1984',   3, 1.00, 3   ),
  (5,  'Ján',       'Polienko', 'muž',  '14.4.1982',  5, 2.28, 1   ),
  (6,  'Juraj',     'Truľo',    'muž',  '16.7.1979',  1, 3.00, 1   ),
  (7,  'Jana',      'Botková',  'žena', '21.9.1977',  4, 1.50, NULL),
  (8,  'Dana',      'Botková',  'žena', '21.9.1977',  4, 1.40, NULL),
  (9,  'Ján',       'Hlúpy',    'muž',  '1.4.1972',   2, 3.00, 1   ),
  (10, 'Aladár',    'Miazga',   'muž',  '22.12.1987', 3, 2.06, NULL),
  (11, 'Mikuláš',   'Myšiak',   'muž',  '6.6.1983',   5, 1.66, 4   ),
  (12, 'Donald',    'Káčer',    'muž',  '7.10.1982',  5, 1.83, 4   ),
  (13, 'Jozef',     'Námorník', 'muž',  '23.9.1981',  2, 2.90, NULL),
  (14, 'Peter',     'Pan',      'muž',  '13.1.2001',  1, NULL, 5   )
```

Žiaľ, v súčasnosti nie je v DB2 z tabuľky možné mazať existujúce stĺpce. Malou náplasťou je možnosť zväčšiť rozsah stĺpcov, ale iba dátového typu `VARCHAR`. Ak napríklad chceme umožniť študovať Pippi Dlhej Pančuche, plným menom Pippilotta Viktuália Roleta Zlatka Efraimová Pančušisková Dlhá Pančucha, musíme zväčšiť dátový rozsah stĺpcov `meno` a `priezvisko` aspoň na takúto hodnotu:

```
ALTER TABLE študent ALTER COLUMN meno SET DATA TYPE VARCHAR(34)
;
ALTER TABLE študent ALTER priezvisko SET DATA TYPE VARCHAR(36)
```

Po mene tabuľky nasleduje ALTER COLUMN („zmeň stĺpec“), resp. ekvivalentne a skrátene ALTER („zmeň“), doplnené názvom modifikovaného stĺpca, za ktorým je slovné spojenie SET DATA TYPE („nastav dátový typ“) a dátový typ VARCHAR (spolu s rozsahom). Až teraz môžeme vložiť záznam o Pippi:

```
INSERT INTO študent
VALUES
(15,'Pippilotta Viktuália Roleta Zlatka','Efraimová Pančušisková Dlhá Pančucha','žena','23.5.1944',1,NULL,NULL)
```

2.3.2 Pridávanie a mazanie integritných obmedzení

Ďalšou možnosťou modifikovať tabuľku je pridávanie integritných obmedzení – primárny či sekundárny kľúč, kontrola alebo cudzí kľúč. Ako pars pro toto uveďme známu kontrolu študijného priemeru:

```
ALTER TABLE študent ADD CHECK (priemer BETWEEN 1 AND 3)
```

Po názve tabuľky (tak ako pri stĺpcoch) nasleduje slovo ADD, tentoraz je však za ním definícia integritného obmedzenia.

Integritné obmedzenia je možné (na rozdiel od stĺpcov) dokonca aj mazať. Primárneho kľúča sa zbavíme jednoducho:

```
ALTER TABLE študent DROP PRIMARY KEY
```

Po mene tabuľky teraz nasleduje už známe mazacie slovo DROP, slovné spojenie PRIMARY KEY za ním hovorí samo za seba. S ostatnými integritnými obmedzeniami je to ťažšie – na rozdiel od primárneho kľúča ich môže byť v jednej tabuľke viac, a preto sa ťažšie identifikujú. Jedným z riešení tejto technickej komplikácie je prideliť každému obmedzeniu (azda okrem primárneho kľúča) nejaký názov, a to už pri jeho definícii – tak, že predeň napíšeme slovo CONSTRAINT („obmedzenie“). (V skutočnosti má integritné obmedzenie názov vždy – ak to neurobíme my explicitne, pridelí mu ho systém (je v ňom zakódovaný okamih jeho vytvorenia, vyzerá však dosť hrozne).)

Takéto pomenovanie sme mohli urobiť už aj v definícii tabuľky. Napríklad v oboch nasledujúcich (ekvivalentných) vyjadreniach sme nazvali f_bydlisko cudzí kľúč na tabuľku krajina a c_pohlavie kontrolu pohlavia:

```
CREATE TABLE študent_c
(
  id              INT         NOT NULL PRIMARY KEY,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4)     NOT NULL,
  dátum_narodenia DATE        NOT NULL,
  ročník          INT         NOT NULL,
  priemer         DEC(3,2),
  id_bydlisko     INT,
  CONSTRAINT f_bydlisko FOREIGN KEY (id_bydlisko) REFERENCES krajina,
  CONSTRAINT c_pohlavie CHECK (pohlavie IN ('muž', 'žena'))
)
```

alebo troma príkazmi:

```
CREATE TABLE študent_c
(
  id              INT         NOT NULL PRIMARY KEY,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4)     NOT NULL,
  dátum_narodenia DATE        NOT NULL,
  ročník          INT         NOT NULL,
  priemer         DEC(3,2),
  id_bydlisko     INT
)
;
ALTER TABLE študent_c ADD CONSTRAINT f_bydlisko FOREIGN KEY (id_bydlisko) REFERENCES krajina
;
ALTER TABLE študent_c ADD CONSTRAINT c_pohlavie CHECK (pohlavie IN ('muž', 'žena'))
```

Teraz už nie je problém tieto obmedzenia vymazať:

```
ALTER TABLE študent_c DROP FOREIGN KEY f_bydlisko
;
ALTER TABLE študent_c DROP CHECK c_pohlavie
```

alebo jednotnejšou formou:

```
ALTER TABLE študent_c DROP CONSTRAINT f_bydlisko
;
ALTER TABLE študent_c DROP CONSTRAINT c_pohlavie
```

V prvom prípade teda po slove `DROP` nasleduje druh mazaného integritného obmedzenia a jeho názov, v druhom slovo `CONSTRAINT` a taktiež názov obmedzenia. Z týchto vyjadrení vyplýva, že **názvy integritných obmedzení v jednej tabuľke musia byť bez ohľadu na ich druh rôzne**.

A na záver ešte jedna malá zaujímavosť: Predpokladajme, že sme pri definícii tabuľky pri niektorom stĺpci použili obmedzenie `NOT NULL`. Ak nejde o kľúčový stĺpec (kde je toto obmedzenie nutné), po čase môže vyvstať pôvodne nepredvídaná potreba ukladať doň aj prázdne hodnoty. Vtedy je už, žiaľ, neskoro, pretože tohto obmedzenia sa už nevieme zbaviť. Keby sme však pri definícii namiesto neho použili kontrolu hovoriacu, že stĺpec „`IS NOT NULL`“, v prípade takejto potreby ju jednoducho zrušíme. Kvôli jednoduchosti však aj v prípade nekľúčových stĺpcov ostaneme pri `NOT NULL`.

2.4 Návrh väčšej databázovej štruktúry

2.4.1 Konceptuálne modelovanie

Skúsenosti, ktoré sme už získali, nás oprávňujú na expanziu nášho doterajšieho databázového systému. Pokúsme sa preto komplexnejšie rozobrať to, ako to na našej univerzite chodí. Nebudú nás zaujímať len študenti, ale aj učitelia a predmety. U študentov sme si doteraz všímali meno a priezvisko, pohlavie, dátum narodenia, bydlisko, ročník a študijný priemer. Ďalšími zaujímavými vlastnosťami by boli študijná skupina, predmety, ktoré si študent zapísal, (aj s hodnotením, ak už existuje) a izba, kde je ubytovaný (univerzita má svoj jediný internát). U učiteľov by to tiež bolo, samozrejme, meno a priezvisko, pohlavie a dátum narodenia. Na rozdiel od študentov nás tu však nebude zaujímať krajina pôvodu, zato titul áno. Ročník, študijná skupina a študijné výsledky tu nemajú vôbec zmysel. Navyše sú niektorí učitelia a študenti členmi školskej rady. Predmety sú tu tiež zaujímavé, ale v obrátenom garde – tu pôjde o to, ktorý predmet učiteľ učí. Pri predmetoch samotných nás bude zaujímať ich názov a skratka, počet kreditov, to, aké sú medzi nimi vzájomné vzťahy (prerekvizity), ktorí študenti si ich zapísali a aké známky z nich dostali, a aj to, ktorý učiteľ ich vyučuje (povedzme, že vzhľadom na malosť našej univerzity je každý predmet vyučovaný jediným učiteľom). Prepíšme si tieto požiadavky do prehľadnejšej formy:

Študent:

- meno
- priezvisko
- dátum narodenia
- pohlavie
- bydlisko
- študijná skupina
- ročník
- zapísané predmety (pri každom aj hodnotenie)
- študijný priemer
- číslo izby
- členstvo v rade

Učiteľ:

- meno
- priezvisko
- dátum narodenia
- pohlavie
- titul
- vyučované predmety
- členstvo v rade

Predmet:

- skratka
- názov
- počet kreditov

- učiteľ
- zapísaní študenti (u každého aj hodnotenie)
- prerekvizity

Troška filozofie nezaškodí: Každý konkrétny študent (napr. Ružena Šípová) je tzv. **entitou** (alebo **súcnom**) – samostatne existujúcim a identifikovateľným jednotlivcom, **objektom**. Keďže u všetkých študentov nás zaujímajú rovnaké charakteristické znaky čiže atribúty (i keď nemusia mať rovnaké hodnoty), títo študenti tvoria jeden **entitný typ**. Jeho prirodzený názov je zrejme *Študent*, tu však nemyslíme žiadneho konkrétneho jednotlivca, ale **pojem** študenta, akúsi jeho **platónovskú ideu**. Každý jeden konkrétny študent je potom jej **inštanciou** – zhmotnením, objektom s konkrétnymi hodnotami abstraktných atribútov.

Rozdiel medzi platónovskými ideami a ich inštanciami dobre vidieť v jazykoch, ktoré majú určitý a neurčitý člen. Trebárs v angličtine si pod „*a* man“ (teda s **neurčitým členom**) predstavíme akéhosi neurčitého človeka, resp. človeka „ako takého“. Nedá sa pritom určiť napríklad konkrétna farba jeho očí či jeho výška, vieme iba, že takúto vlastnosť určite má. Naproti tomu „*the* man“ (teda s **určitým členom**) znamená konkrétneho človeka, o ktorom by sme (možno pri troške snahy) tieto vlastnosti dokázali určiť celkom presne. Všimnime si, že vyjadrenie s neurčitým členom zodpovedá nášmu entitnému typu – pri ňom tiež nepoznáme konkrétne hodnoty atribútov (hoci niektoré môžeme predpokladať). „A man“ je teda vlastne **množina** ľudí, a keď povieme „John is a man.“, znamená to, že John je **prvkom** tejto množiny. Vyjadrenie s určitým členom „John is the man.“ zas znamená informáciu, že ten človek, na ktorého spoločne myslia účastníci rozhovoru (teda jeden úplne konkrétny človek), sa volá John.

S platónovskými ideami sa však stretávame v každom jazyku, bez ohľadu na to, či má členy. Ak povieme vetu „Človek má na ruke päť prstov.“, vyslovujeme tým akési **pravidlo**. To však neznamená, že postihnutí polydaktýliou alebo nešikovní pracovníci píly nie sú ľudia. Máme totiž na mysli práve túto platónovskú ideu človeka s preddefinovaným (ale zmeniteľným) počtom prstov. Podobne pravidlo „Slon africký žije v Afrike.“ v žiadnom prípade nevylučuje z príslušnosti k svojmu rodu najväčšie (v oboch slova zmysloch) atrakcie bojnickej ZOO, lebo opäť máme na mysli slona platónovského. Naša reč je teda plná vyjadrení o platónovských ideách, formulované pravidlá však často pripúšťajú výnimky. **Ak by sme mali vždy hovoriť o konkrétnych inštanciách, naša reč by možno bola presnejšia, ale nedokázala by veľkoryso prehliadnuť (často nepodstatné) odlišnosti, paralyzovala by tak schopnosť zovšeobecňovať a znemožňovala abstraktné myslenie.**

Táto filozofia sa s úspechom uplatňuje aj v informatike. **Podstatou objektového programovania je uvedomenie si základných konceptov – tried, ktoré zodpovedajú platónovským ideám, a objektov, ktoré sú ich zhmotneniami.** Z databázového hľadiska **je entitný typ spravidla vhodným kandidátom na tabuľku** a (ako sme už mali možnosť vidieť v predchádzajúcich statiach) **každej jeho entite zodpovedá práve jeden jej riadok**. Atribúty korešpondujú so stĺpcami tejto tabuľky a v jej políčkach sú konkrétne hodnoty príslušných atribútov.

Doterajšie pozorovania môžeme zhrnúť do nasledujúcej tabuľky:

entitný typ	**entita**
platónovská idea	inštancia
množina	prvok
neurčitý člen	určitý člen
trieda	objekt
tabuľka	záznam

Vytypovali sme teda zatiaľ entitné typy *Študent*, *Učiteľ* a *Predmet*, podriadené odrážky naznačujú ich možné atribúty. Uvedomme si však, že pri skoro akejkoľvek voľbe atribútov by sa mohlo stať, že by sa nám niektoré záznamy nepodarilo odlíšiť (pamätáte sa na Jánov Hlúpych?). Aby sme sa tomu vyhli, je vhodné všetky atribúty zhrnúť do jedného umelého – čísla, ktoré (ak sa to aj nepodarilo žiadnemu inému atribútu) určite dva (skoro) rovnaké záznamy predsa len rozlíši. Každý objekt tak bude mať svoj **identifikátor**, ktorý bude (primárnym) kľúčom v príslušnej tabuľke.

Vidíme, že niektoré požiadavky sú zapísané z dvoch rôznych pohľadov, čo naznačuje interakciu, vzťah, alebo reláciu týchto entitných typov. Najjednoduchším prípadom je vzťah predmetov a učiteľov. Ako v budúcej tabuľke registrovať k jednému učiteľovi všetky ním vyučované predmety? Máme vcelku tri možnosti:

1 Do políčka tabuľky na priesečníku riadku učiteľa a stĺpca reprezentujúceho predmety vložíme viacero predmetov. Aký typ však bude mať výsledok? Celé číslo už asi ťažko, museli by byť uložené v jednom reťazci, ktorý by bolo zakaždým treba syntakticky analyzovať a zisťovať jeho jednotlivé zložky. Vieme si predstaviť, ako by sme sa odvolávali napr. na druhý z učiteľových troch predmetov? No hrôza!...

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	TITUL	ID_PREDMET-y (???)
1	Gejza	Miazga	muž	12.12.1955	.doc	2, 3, 8
2	Matej	Múdry	muž	11.6.1945	.doc	11, 12
3	Vasilisa	Premúdra	žena	22.1.1973	Mgr.	7, 9
4	Hedviga	Baba	žena	3.5.1784	Mgr.	1, 6
5	d'Eduard	Vševed	muž	13.3.1900	DEd	4, 5, 10

2 Jednému učiteľovi bude v budúcej tabuľke zodpovedať toľko riadkov, koľko predmetov vyučuje – každý z nich bude v osobitnom riadku. To by však znamenalo, že všetky ostatné hodnoty atribútov učiteľa by sa v každom z týchto riadkov opakovali, a bolo by to tak porušenie zásady neopakovateľnosti údajov. Navyše by to odporovalo prirodzenej myšlienke, že jednému učiteľovi má v tabuľke s názvom učiteľ zodpovedať jediný riadok. Táto možnosť teda tiež nevyhovuje.

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	TITUL	ID_PREDMET
1	Gejza	Miazga	muž	12.12.1955	.doc	2
1	Gejza	Miazga	muž	12.12.1955	.doc	3
1	Gejza	Miazga	muž	12.12.1955	.doc	8
2	Matej	Múdry	muž	11.6.1945	.doc	11
2	Matej	Múdry	muž	11.6.1945	.doc	12
3	Vasilisa	Premúdra	žena	22.1.1973	Mgr.	7
3	Vasilisa	Premúdra	žena	22.1.1973	Mgr.	9
4	Hedviga	Baba	žena	3.5.1784	Mgr.	1
4	Hedviga	Baba	žena	3.5.1784	Mgr.	6
5	d'Eduard	Vševed	muž	13.3.1900	DEd	4
5	d'Eduard	Vševed	muž	13.3.1900	DEd	5
5	d'Eduard	Vševed	muž	13.3.1900	DEd	10

3 Pre každý z vyučovaných predmetov vymedzíme osobitný stĺpec. To by sme však najprv museli odhadnúť, koľko predmetov učiteľ učí. Ak budeme príliš šetriť, a toto číslo bude malé, nemali by sme kde umiestniť učiteľa, ktorý ich učí čo len o jeden viac, než sme navrhli. Ak zasa budeme príliš veľkorysí, pri drvivej väčšine záznamov nebudú tieto stĺpce vôbec využité – uvedomme si, že stĺpce navrhujeme pre tabuľku ako celok, nie pre každý záznam osobitne. Navyše by sme pri neskoršom hľadaní predmetu museli prehľadávať všetky tieto stĺpce. Už len jednoduchá požiadavka zistiť k učiteľovi všetky jeho predmety by znamenalo nesmierne technické problémy, o zložitejších ani nevraviac. Ani takto to teda nepôjde.

ID	MENO	PRIEZVISKO	...	ID_PREDMET1	ID_PREDMET2	ID_PREDMET3	ID_PREDMET4	...
1	Gejza	Miazga	...	2	3	8	NULL	...
2	Matej	Múdry	...	11	12	NULL	NULL	...
3	Vasilisa	Premúdra	...	7	9	NULL	NULL	...
4	Hedviga	Baba	...	1	6	NULL	NULL	...
5	d'Eduard	Vševed	...	4	5	10	NULL	...

Takže kam z konopí? Skúsme sa na to pozrieť zo strany predmetov. Vzhľadom na to, že ku každému predmetu je priradený jediný učiteľ, tu vyššie uvedené problémy odpadávajú – informácia o tom, ktorý učiteľ predmet vyučuje, je elementárna, a môžeme ju teda vložiť do príslušného políčka. Od tej chvíle už informácia o tom, ktorý učiteľ vyučuje ktorý predmet, v databáze je (a to v tabuľke predmet), a nemusíme (ba nesmieme) ju už (aj vzhľadom na minimalizáciu redundancie) dávať aj do tabuľky učiteľ. Takto môžeme vyjadriť ľubovoľnú **reláciu typu 1:n**, čo znamená, že platí:

1 Každej inštancii prvého entitného typu zodpovedá v princípe niekoľko inštancií druhého entitného typu.

2 Každej inštancii druhého entitného typu zodpovedá v princípe (najviac) jedna inštancia prvého entitného typu.

Takúto reláciu potom budeme reprezentovať cudzím kľúčom, tabuľka prvého entitného typu bude odkazovanou a tabuľka druhého odkazujúcou (tu bude definovaný ten cudzí kľúč). V našom prípade je prvým entitným typom *Učiteľ* a druhým *Predmet*, cudzí kľúč teda bude definovaný v tabuľke `predmet`:

ID	KÓD	NÁZOV	KREDIT	ID_UČITEĽ
1	UPR	Umenie pomáhať rozhovorom	3	4
2	VZS	Sociálnopsychologický výcvik zvládania záťažových situácií	2	1
3	NEM	Nebeská mechanika	4	1
4	ZLO	Zložité systémy	9	5
5	HNR	Hlbokonepružný rozptyl leptónov na hadrónoch	8	5
6	MMO	Molekulový modeling	3	4
7	ROM	Romológia	6	3
8	HRO	Teória hromadnej obsluhy	4	1
9	HRY	Teória hier	3	3
10	FYD	Fyzika DNA	3	5
11	FAK	Fázové prechody a kritické javy	5	2
12	DBS	Databázové systémy	5	2

a bude ukazovať na tabuľku `učiteľ`:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	TITUL
1	Gejza	Miazga	muž	12.12.1955	.doc
2	Matej	Múdry	muž	11.6.1945	.doc
3	Vasilisa	Premúdra	žena	22.1.1973	Mgr.
4	Hedviga	Baba	žena	3.5.1784	Mgr.
5	d'Eduard	Vševed	muž	13.3.1900	DEd

Skúsme toto šalamúnske riešenie použiť aj v prípade relácie medzi entitnými typmi *Študent* a *Predmet*. Tu je však situácia trochu zložitejšia. Ani v jednom smere totiž nejde o reláciu typu 1:n. Platí totiž:

1 Každej inštancii prvého entitného typu zodpovedá v princípe niekoľko inštancií druhého entitného typu.

2 Každej inštancii druhého entitného typu zodpovedá v princípe niekoľko inštancií prvého entitného typu.

O takej relácii hovoríme, že je **typu m:n**. Vtipnosť predchádzajúceho riešenia spočívala v tom, že sme problematický atribút z problematickej tabuľky jednoducho vynechali a zodpovednosť zvalili na iný entitný typ, ktorý sa jej vedel pohodlne zhostiť. Na koho však preniesť túto úlohu teraz, keď oba entitné typy sú v rovnakých trapiech ako minule *Učiteľ*? Každý predmet totiž môže mať zapísaný viacero študentov a každý študent môže mať zapísaných viacero predmetov (dokonca (v prípade neúspechu) aj viackrát). Prirodzená odpoveď je, že by to mal zabezpečiť niekto iný, tretia strana, sprostredkovateľ. Vytvoríme preto ďalší (i keď možno na prvý pohľad trochu umelý) entitný typ s názvom *Zapísané* (chceme totiž vystihnúť fakt, že nejaký študent si zapísal nejaký predmet). Atribútmi tohto entitného typu potom určite budú *študent* a *predmet*, núkajú sa však aj ďalšie – *dátum zápisu*, *dátum (úspešného alebo neúspešného) ukončenia* a *hodnotenie* (prípustné hodnoty sú `1`, `2`, `3`, alebo `NULL` (ak študent ešte predmet neabsolvoval)). Všimnime si, že nové relácie (jedna medzi entitnými typmi *Študent* a *Zapísané* a druhá medzi entitnými typmi *Predmet* a *Zapísané*) sú obe typu 1:n – každému študentovi prislúcha niekoľko zápisov predmetu, každému predmetu tiež prislúcha niekoľko zápisov od študentov, ale každý zápis predmetu sa týka práve jedného študenta a práve jedného predmetu. Relácia medzi entitnými typmi *Študent* a *Predmet* tak stratila zmysel – túto reláciu typu m:n sme plne nahradili dvoma novými reláciami typu 1:n. Takýto proces nazývame **dekompozícia relácie typu m:n**.

Teraz sa už budeme vedieť vysporiadať aj s prerekvizitami predmetov. Rozumieme pod nimi to, že absolvovanie jedného predmetu musí predchádzať zápisu (alebo absolvovaniu?) iného predmetu. Každá jednotlivá prerekvizita je teda vzťah medzi dvoma konkrétnymi predmetmi, čiže dvoma inštanciami entitného typu *Predmet*. Každý predmet môže byť prerekvizitou mnohých iných predmetov, ale môže ich mať niekoľko aj sám. Ide teda o reláciu typu m:n, ktorú budeme dekomponovať podľa predchádzajúceho postupu. (Jedinou novinkou je, že na oboch stranách relácie je ten istý entitný typ, to však na veci nič nemení.) Vznikne tak nový entitný typ, ktorý môžeme nazvať celkom prirodzene *Prerekvizita* a ktorého atribúty sú dva predmety – *podmieňujúci* a *podmieňovaný*. Entitný typ *Predmet*, na ktorý sa tieto dva atribúty odvolávajú, tak vystupuje v dvoch **rolách** – raz ako podmieňujúci, raz ako podmieňovaný. Vzhľadom na to, že žiadnu dvojicu predmetov v novom entitnom type *Prerekvizita* nemá zmysel opakovať, pár atribútov *podmieňujúci* a *podmieňovaný* je vhodným kandidátom na identifikátor (nemusíme teda pridávať umelý).

Všimnime si teraz podobnosť i rozdielnosť entitných typov *Študent* a *Učiteľ*. Hoci majú niektoré atribúty odlišné, predsa však obe hovoria o nejakých osobách, ktoré môžeme charakterizovať tými istými znakmi – menom, priezviskom, pohlavím a dátumom narodenia. Stojí preto za to vytvoriť nový entitný typ *Osoba*, ktorý bude tieto spoločné atribúty združovať. Avšak vzhľadom na rozdielnosť zvyšných atribútov na pôvodné entitné typy *Študent* a *Učiteľ* nemôžeme rezignovať, v budúcej tabuľke `osoba` by to znamenalo priveľa principiálne prázdnych hodnôt – pri záznamoch učiteľov v čisto študentských atribútoch (napr. *ročník* či *študijná skupina*) a pri záznamoch študentov naopak v čisto učiteľských stĺpcoch (*titul*). Každá entita typu *Študent* či *Učiteľ* teda bude namiesto spoločných atribútov obsahovať len odkaz na príslušnú abstraktnejšiu entitu typu *Osoba*. Rečou objektového programovania teda môžeme povedať, že entitné typy *Študent* aj *Učiteľ* sú **dedičmi** entitného typu *Osoba*, sú jeho **špecializáciami**, a on je naopak ich **generalizáciou**, **zovšeobecnením**. Všetky tri entitné typy tvoria tzv. **IS-A hierarchiu** (z anglického „X is a Y“ – „X je (nejaké) Y“), lebo študent je (nejaká) osoba a učiteľ je tiež (nejaká) osoba. Relácie medzi entitnými typmi *Študent* a *Osoba* a tiež medzi *Učiteľ* a *Osoba* sú **relácie typu 1:1**, čo znamená, že platí:

1. Každej inštancii prvého entitného typu zodpovedá v princípe (najviac) jedna inštancia druhého.
2. Každej inštancii druhého entitného typu zodpovedá v princípe (najviac) jedna inštancia prvého.

Všimnime si teraz, ako je na tom ďalší kandidát na entitný typ – *Člen rady*. Najprv si uvedomme, že by to naozaj mal byť osobitný entitný typ, jeho špeciálne atribúty nestačí len priradiť k už dohodnutým entitným typom. Keďže členstvo v rade sa môže týkať učiteľa aj študenta, určite by nestačilo rozšíriť ani jeden z entitných typov *Študent* alebo *Učiteľ*. Typu *Osoba* by tieto príliš špeciálne atribúty tiež nesvedčali – *váha hlasu* jednoducho nie je vlastnosťou osoby ako takej. Teda entitný typ *Člen rady* bude špecializáciou typu *Osoba*, tak ako entitné typy *Študent* a *Učiteľ*. Je tu však jeden dôležitý rozdiel. Hoci sú všetky tri spomínané entitné typy dedičmi typu *Osoba*, predsa len dvojica *Študent* a *Učiteľ* tvorí pevnejšiu väzbu: Vzhľadom na to, že každá osoba musí byť (v našom modeli) buď študentom, alebo učiteľom, typ *Osoba* je ich zoskupením – **úplnou a disjunktnou generalizáciou**. S dedičnosťou typu *Člen rady* voči typu *Osoba* je to inak, tu sa o úplnosti a disjunktnosti hovoriť nedá.

Pozrime sa teraz na ostatné relácie: Problém s krajinou pôvodu sme už raz vyriešili – vytvoríme nový entitný typ *Krajina*, na ktorý bude ukazovať entitný typ *Študent*. Dodajme len, že táto relácia je typu 1:n a názov krajiny by mal byť jednoznačný. Analogicky je to so študijnou skupinou (črtá sa entitný typ *Študijná skupina* s atribútmi (jednoznačná) *skratka* a (jednoznačný) plný *názov*) a s izbou (entitný typ *Izba* s atribútom (jednoznačné) *číslo*). Pri entitnom type *Učiteľ* rovnako vyriešime ďalšiu reláciu typu 1:n, a to vzťah k titulu (entitný typ *Titul* s atribútmi (jednoznačná) *skratka* a (jednoznačný) plný *názov*), a analogicky pri type *Člen rady* vzťah k funkcii (entitný typ *Funkcia v rade* s atribútom (jednoznačný) *názov*). Rovnako by sme mohli postupovať aj v prípade entitného typu *Osoba* a atribútu *pohlavie*, ale vzhľadom na to, že sú prípustné principiálne len dve hodnoty (všimnime si, že v ostatných prípadoch sa môže počet entít daného typu meniť), je vhodnejšie nový entitný typ nepridávať a *pohlavie* ponechať ako atribút entitného typu *Osoba*.

Rozoberme ešte problém ubytovania. V (jedinom) univerzitnom internáte sú izby niekoľkých druhov, ktoré sa okrem iného líšia zariadením, počtom lôžok a mesačným poplatkom. Ak budeme chcieť zisťovať sumu vybraných mesačných poplatkov, bude vhodné doplniť ďalší entitný typ – *Druh izby*, kde bude okrem (jedno-

značného) atribútu *kód* ešte *kapacita*, *mesačný poplatok* a *popis* – informáciu o typickom zariadení. Navyše do entitného typu *Izba* dodáme odkaz na jej druh.

Posledným problémom je *študijný priemer* – predpokladaný atribút entitného typu *Študent*. Uvedomme si, že teraz je už nadbytočný, môžeme ho totiž vypočítať z hodnotení doteraz absolvovaných predmetov. Nie je síce úplným nezmyslom si ho ponechať (ak máme pocit, že by sme ho mohli často či rýchlo potrebovať) a neodporuje to ani požiadavke neduplicity dát (keďže ide o odvodený údaj, ktorý sa v takejto forme v databáze nevyskytuje), ale museli by sme nejako zabezpečiť, že jeho hodnota bude v každom okamihu v súlade s príslušnými hodnoteniami. Bude teda bezpečnejšie a jednoduchšie tento odvodený atribút vynechať.

Končí sa **konceptuálne modelovanie** (vybranej časti) situácie na našej univerzite (lepšie povedané jeho prvá iterácia, lebo dodatočné požiadavky alebo praktické skúsenosti v ňom môžu spôsobiť menšie či väčšie zmeny). Vytypovali sme entitné typy a určili **kardinalitu** vzťahov medzi nimi (t. j. to, či ide o relácie typu 1:1, 1:n, alebo m:n), vytvorili sme tak **entitno-relačný model** reálnej situácie. Zhrňme si ho (v zátvorkách sú niektoré dodatočné informácie):

Entitné typy:

- *Osoba*:
  - *identifikátor*
  - *meno*
  - *priezvisko*
  - *pohlavie* (možné hodnoty iba `muž` alebo `žena`)
  - *dátum narodenia*
- *Študent*:
  - *identifikátor* (odkaz na entitný typ *Osoba*)
  - *bydlisko* (odkaz na entitný typ *Krajina*)
  - *študijná skupina* (odkaz na entitný typ *Študijná skupina*)
  - *ročník* (možné hodnoty iba z intervalu `1` až `5`, predvolená hodnota je `1`)
  - *izba* (odkaz na entitný typ *Izba*)
- *Učiteľ*:
  - *identifikátor* (odkaz na entitný typ *Osoba*)
  - *titul* (odkaz na entitný typ *Titul*)
- *Člen rady*:
  - *identifikátor* (odkaz na entitný typ *Osoba*)
  - *funkcia* (odkaz na entitný typ *Funkcia v rade*)
  - *váha hlasu* (predvolená hodnota je `1`)
- *Funkcia v rade*:
  - *identifikátor*
  - *názov* (jednoznačný)
- *Predmet*:
  - *identifikátor*
  - *kód* (jednoznačný)
  - *názov* (jednoznačný)

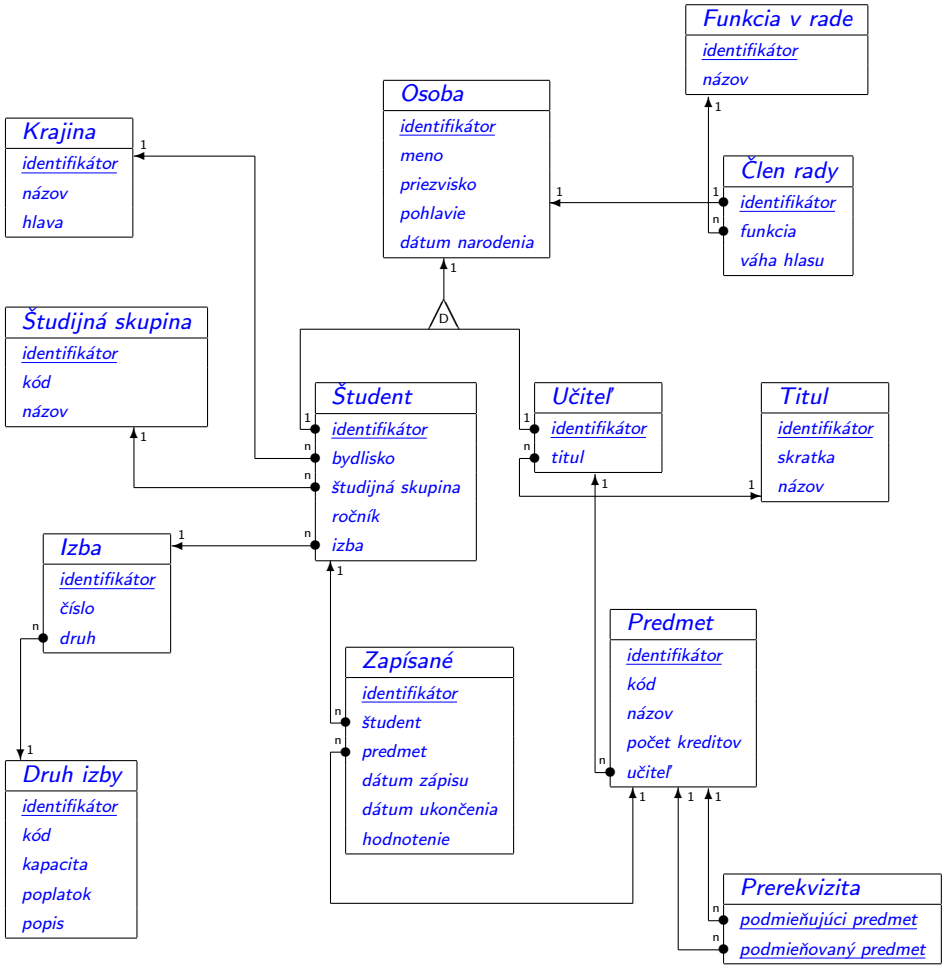
  - *počet kreditov*
  - *učiteľ* (odkaz na entitný typ *Učiteľ*)
- *Prerekvizita* (identifikátor je tvorený jedinými dvoma atribútmi):
  - *podmieňujúci predmet* (odkaz na entitný typ *Predmet*)
  - *podmieňovaný predmet* (odkaz na entitný typ *Predmet*)
- *Zapísané*:
  - *identifikátor*
  - *študent* (odkaz na entitný typ *Študent*)
  - *predmet* (odkaz na entitný typ *Predmet*)
  - *dátum zápisu* (predvolená hodnota je aktuálny deň)
  - *dátum (úspešného alebo neúspešného) ukončenia*
  - *hodnotenie* (možné hodnoty iba `1`, `2`, `3`, alebo prázdna)
- *Študijná skupina*:
  - *identifikátor*
  - *kód* (jednoznačný)
  - *názov* (jednoznačný)
- *Krajina*:
  - *identifikátor*
  - *názov* (jednoznačný)
  - *hlava*
- *Titul*:
  - *identifikátor*
  - *skratka* (jednoznačná)
  - *názov* (jednoznačný)
- *Izba*:
  - *identifikátor*
  - *číslo* (jednoznačné)
  - *druh* (odkaz na entitný typ *Druh izby*)
- *Druh izby*:
  - *identifikátor*
  - *kód* (jednoznačný)
  - *kapacita*
  - *poplatok*
  - *popis*

Vzťahy (včítane kardinality):

- *Osoba* – *Študent*: typu 1:1 (generalizácia)
- *Osoba* – *Učiteľ*: typu 1:1 (generalizácia)

- *Osoba* – *Člen rady*: typu 1:1
- *Krajina* – *Študent*: typu 1:n, prázdne hodnoty odkazu prípustné
- *Študijná skupina* – *Študent*: typu 1:n
- *Izba* – *Študent*: typu 1:n, prázdne hodnoty odkazu prípustné
- *Titul* – *Učiteľ*: typu 1:n
- *Funkcia v rade* – *Člen rady*: typu 1:n
- *Učiteľ* – *Predmet*: typu 1:n
- *Predmet* – *Prerekvizita*: typu 1:n, rola *podmieňujúci*
- *Predmet* – *Prerekvizita*: typu 1:n, rola *podmieňovaný*
- *Študent* – *Zapísané*: typu 1:n
- *Predmet* – *Zapísané*: typu 1:n
- *Druh izby* – *Izba*: typu 1:n

Graficky to bude azda prehľadnejšie:



Náš model sme teda rozšírili na 13 tabuliek. Pravdaže, toto číslo nemusí byť konečné, nikto predsa nikdy netvrdil, že sme problematiku našej univerzity úplne vyčerpali. V ďalšej verzii by sme možno mohli uvažovať trebárs o možnosti vytvárať rozvrh hodín (čo by vyžadovalo udržiavať aj informácie o miestnostiach a časoch prednášok) či o zlepšení evidencie skúšania (zatiaľ sme neuvažovali o poradí termínov skúšok). Ak sa však chceme dostať ďalej, zákonite musíme niekde proces rozširovania useknúť. Tu.

2.4.2 Transformácia modelu do databázy

Vytvorme teda všetky tabuľky databázy od piky. Vieme už, že každá z nich zodpovedá jednému entitnému typu a každý jej stĺpec niektorému atribútu. Ak je atribút odkazom, premietne sa do cudzieho kľúča. Doplnkové informácie v zátvorkách budú korešpondovať s ďalšími integritnými obmedzeniami (`CHECK` a `UNIQUE`) alebo s preddefinovanou hodnotou (`DEFAULT`). Všimnime si tiež, že okrem prípadov, kde na to máme vážny dôvod, všetky ostatné stĺpce nepripúšťajú prázdne hodnoty. O tom, že každá tabuľka má primárny kľúč, sa už hádam zmieňovať netreba.

Každé integritné obmedzenie (okrem primárneho kľúča) pomenujeme, a aby sme nemuseli dávať pozor na poradie vytvárania tabuliek, cudzie kľúče vytvoríme pomocou `ALTER TABLE`:

```
CREATE TABLE osoba
(
  id              INT         NOT NULL,
  meno            VARCHAR(10) NOT NULL,
  priezvisko      VARCHAR(15) NOT NULL,
  pohlavie        CHAR(4)     NOT NULL,
  dátum_narodenia DATE        NOT NULL,
  PRIMARY KEY (id),
  CONSTRAINT c_osoba_pohlavie CHECK (pohlavie IN ('muž','žena'))
)
;
CREATE TABLE učiteľ
(
  id              INT         NOT NULL,
  id_titul        INT         NOT NULL,
  PRIMARY KEY (id)
)
;
CREATE TABLE titul
(
  id              INT         NOT NULL,
  skratka         VARCHAR(10) NOT NULL,
  názov           VARCHAR(30) NOT NULL,
  PRIMARY KEY (id),
  CONSTRAINT u_titul_skratka UNIQUE (skratka),
  CONSTRAINT u_titul_názov UNIQUE (názov)
)
;
CREATE TABLE študent
(
  id              INT         NOT NULL,
  id_skupina      INT         NOT NULL,
  ročník          INT         NOT NULL DEFAULT 1,
  id_bydlisko     INT,
  id_izba         INT,
  PRIMARY KEY (id),
  CONSTRAINT c_študent_ročník CHECK (ročník BETWEEN 1 AND 5)
)
;
CREATE TABLE študijná_skupina
(
  id              INT         NOT NULL,
  kód             CHAR(2)     NOT NULL,
  názov           VARCHAR(30) NOT NULL,
  PRIMARY KEY (id),
  CONSTRAINT u_skupina_kód UNIQUE (kód),
  CONSTRAINT u_skupina_názov UNIQUE (názov)
)
;
```

```
CREATE TABLE krajina
(
  id              INT         NOT NULL,
  názov           VARCHAR(50) NOT NULL,
  hlava           VARCHAR(50),
  PRIMARY KEY (id),
  CONSTRAINT u_krajina_názov UNIQUE (názov)
)
;
CREATE TABLE izba
(
  id              INT         NOT NULL,
  číslo           VARCHAR(5)  NOT NULL,
  id_druh         INT         NOT NULL,
  PRIMARY KEY (id),
  CONSTRAINT u_izba_číslo UNIQUE (číslo)
)
;
CREATE TABLE druh_izby
(
  id              INT         NOT NULL,
  kód             CHAR(5)     NOT NULL,
  kapacita        INT         NOT NULL,
  poplatok        DEC(6,2)    NOT NULL,
  popis           VARCHAR(80) NOT NULL,
  PRIMARY KEY (id),
  CONSTRAINT u_druhizby_kód UNIQUE (kód)
)
;
CREATE TABLE člen_rady
(
  id              INT         NOT NULL,
  id_funkcia      INT         NOT NULL,
  váha_hlasu      INT         NOT NULL DEFAULT 1,
  PRIMARY KEY (id)
)
;
CREATE TABLE funkcia_v_rade
(
  id              INT         NOT NULL,
  názov           VARCHAR(30) NOT NULL,
  PRIMARY KEY (id),
  CONSTRAINT u_funkcia_názov UNIQUE (názov)
)
;
CREATE TABLE predmet
(
  id              INT         NOT NULL,
  kód             CHAR(3)     NOT NULL,
  názov           VARCHAR(60) NOT NULL,
  kredit          INT         NOT NULL,
  id_učiteľ       INT         NOT NULL,
  PRIMARY KEY (id),
  CONSTRAINT u_predmet_kód UNIQUE (kód),
  CONSTRAINT u_predmet_názov UNIQUE (názov)
)
;
CREATE TABLE prerekvizita
(
  id_podmieňovaný INT         NOT NULL,
  id_podmieňujúci INT         NOT NULL,
  PRIMARY KEY(id_podmieňovaný, id_podmieňujúci)
)
;
CREATE TABLE zapísané
(
  id              INT         NOT NULL,
  id_študent      INT         NOT NULL,
  id_predmet      INT         NOT NULL,
  dátum_zápisu    DATE        NOT NULL DEFAULT CURRENT DATE,
  dátum_ukončenia DATE,
  hodnotenie      INT,
  PRIMARY KEY (id),
```

```
  CONSTRAINT c_zapísané_hodn CHECK (hodnotenie BETWEEN 1 AND 3)
)
;
ALTER TABLE učiteľ ADD CONSTRAINT f_učiteľ_osoba FOREIGN KEY (id) REFERENCES osoba
  ON DELETE CASCADE
;
ALTER TABLE učiteľ ADD CONSTRAINT f_učiteľ_titul FOREIGN KEY (id_titul) REFERENCES titul
;
ALTER TABLE študent ADD CONSTRAINT f_študent_osoba FOREIGN KEY (id) REFERENCES osoba
  ON DELETE CASCADE
;
ALTER TABLE študent ADD CONSTRAINT f_študent_krajina FOREIGN KEY (id_bydlisko) REFERENCES krajina
  ON DELETE SET NULL
;
ALTER TABLE študent ADD CONSTRAINT f_študent_skupina FOREIGN KEY (id_skupina) REFERENCES študijná_skupina
;
ALTER TABLE študent ADD CONSTRAINT f_študent_izba FOREIGN KEY (id_izba) REFERENCES izba
;
ALTER TABLE člen_rady ADD CONSTRAINT f_členrady_osoba FOREIGN KEY (id) REFERENCES osoba
  ON DELETE CASCADE
;
ALTER TABLE člen_rady ADD CONSTRAINT f_členrady_funkcia FOREIGN KEY (id_funkcia) REFERENCES funkcia_v_rade
;
ALTER TABLE izba ADD CONSTRAINT f_izba_druhizby FOREIGN KEY (id_druh) REFERENCES druh_izby
;
ALTER TABLE predmet ADD CONSTRAINT f_predmet_učiteľ FOREIGN KEY (id_učiteľ) REFERENCES učiteľ
;
ALTER TABLE prerekvizita ADD CONSTRAINT f_pr_podmieňujúci FOREIGN KEY (id_podmieňujúci) REFERENCES predmet
  ON DELETE CASCADE
;
ALTER TABLE prerekvizita ADD CONSTRAINT f_pr_podmieňovaný FOREIGN KEY (id_podmieňovaný) REFERENCES predmet
  ON DELETE CASCADE
;
ALTER TABLE zapísané ADD CONSTRAINT f_zapísané_študent FOREIGN KEY (id_študent) REFERENCES študent
  ON DELETE CASCADE
;
ALTER TABLE zapísané ADD CONSTRAINT f_zapísané_predmet FOREIGN KEY (id_predmet) REFERENCES predmet
  ON DELETE CASCADE
```

Všimnime si, kde sa používa aké mazanie: Ak vymažeme z tabuľky `osoba` nejaký záznam (t. j. príslušná osoba prerušila kontakty s univerzitou), musí sa, samozrejme, jej prislúchajúci záznam odstrániť aj z tabuliek `člen_rady` a `študent`/`učiteľ`. Ak bola táto osoba navyše študentom, odstránia sa aj všetky záznamy o ňou zapísaných predmetoch. A napokon, ak sa ruší predmet, vymažú sa automaticky aj prerekvizity, v ktorých figuroval. Všetky tieto mazania teda budú kaskádové. Možnosť `ON DELETE SET NULL` prichádza do úvahy, pravdaže, len v prípade, keď dotyčný stĺpec pripúšťa prázdne hodnoty. Ide tu teda len o dva cudzie kľúče v tabuľke `študent`, a to `id_bydlisko` a `id_izba`. Pri prvom z nich je to správanie naozaj vhodné – zrušenie krajiny nemusí hneď znamenať vylúčenie jej bývalých občanov, ale ich existencia zas nemôže zrušeniu krajiny zabrániť; jednoducho len prídu o jej občianstvo. Pri izbe je situácia možno trochu odlišná – predtým než chceme záznam o izbe vymazať, by sme mali postihnutých študentov presťahovať do nejakej inej. V žiadnom prípade by sa však nemalo stať, že by zrušenie krajiny či izby mohlo spôsobiť odstránenie jej obyvateľov, takže kaskádové mazanie do úvahy vôbec neprichádza. Podobne odchod učiteľa nemusí hneď znamenať zánik ním učených predmetov. Predtým než odíde, sa musí explicitne rozhodnúť o ich osude – buď sa ich záznamy vymažú, alebo sa presunú inému učiteľovi.

Teraz tabuľky naplníme (pripomeňme, že vzhľadom na existenciu cudzích kľúčov musíme dávať pozor na poradie tabuliek). Za povšimnutie stojí, že pri vkladaní učiteľa d'Eda Vševeda, ktorého meno obsahuje apostrof, musíme tento znak odlíšiť od apostrofu značiaceho koniec reťazca, a to jeho zdvojením:

```
INSERT INTO krajina
VALUES
  (1, 'Za siedmimi horami a siedmimi dolami', 'Drozdia Brada' ),
  (2, 'Kalifát Bagdad',                       'Harún al-Rašíd'),
  (3, 'Mravenisko',                           'Z'             ),
  (4, 'Hollywood',                            'Simba'         ),
  (5, 'Neverland',                            NULL            ),
  (6, 'Haliganda',                            NULL            )
```

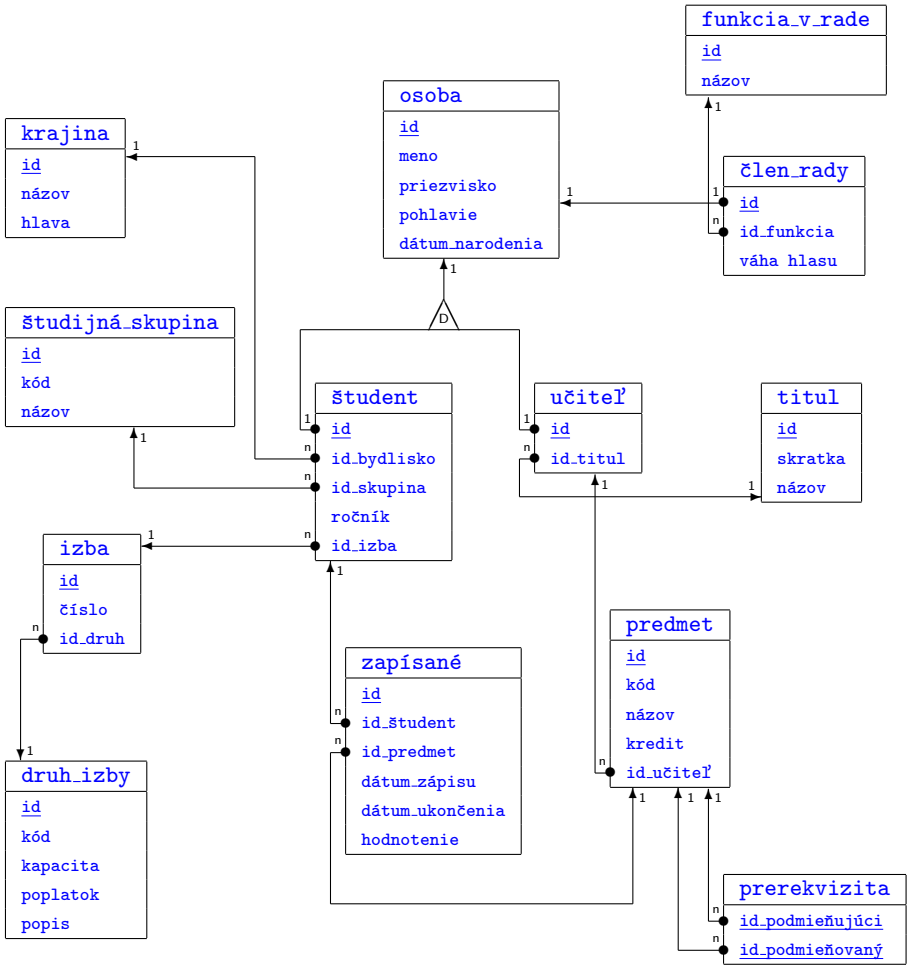
```
;
INSERT INTO titul
VALUES
  (1, '.doc', 'doc(um)ent'         ),
  (2, 'Mgr.', 'managor'            ),
  (3, 'DEd',  'doctor of education')
;
INSERT INTO funkcia_v_rade
VALUES
  (1, 'predseda'            ),
  (2, 'pokladník'           ),
  (3, '(radový) člen'       )
;
INSERT INTO druh_izby
VALUES
  (1, '1A', 1, 1000, 'jednotka s kúpeľňou, chladničkou, TV a internetovou prípojkou'        ),
  (2, '1B', 1,  800, 'jednotka s kúpeľňou, chladničkou a TV, ale bez internetovej prípojky' ),
  (3, '1C', 1,  700, 'jednotka s kúpeľňou a chladničkou, ale bez TV a internetovej prípojky'),
  (4, '2A', 2,  600, 'dvojka s internetovou prípojkou'                                      ),
  (5, '2B', 2,  550, 'dvojka bez internetovej prípojky'                                     ),
  (6, '2P', 3,  400, 'dvojka bez internetovej prípojky s~prístelkou'                        )
;
INSERT INTO izba
VALUES
  (1, '101A', 6),
  (2, '013A', 1),
  (3, '242B', 6),
  (4, '354B', 4),
  (5, '321A', 4),
  (6, '323A', 2)
;
INSERT INTO študijná_skupina
VALUES
  (1, 'M',  'metamatematika'               ),
  (2, 'I',  'informatematika'              ),
  (3, 'MI', 'metamatematika-informatematika')
;
INSERT INTO osoba
VALUES
  (  1, 'Gejza',     'Miazga',   'muž',  '12.12.1955'),
  (  2, 'Matej',     'Múdry',    'muž',  '11.6.1945' ),
  (  3, 'Vasilisa',  'Premúdra', 'žena', '22.1.1973' ),
  (  4, 'Hedviga',   'Baba',     'žena',  '3.5.1784' ),
  (  5, 'd''Eduard', 'Vševed',   'muž',  '13.3.1900' ),
  (101, 'Ján',       'Hraško',   'muž',  '12.7.1987' ),
  (102, 'Ružena',    'Šípová',   'žena', '1.2.1984'  ),
  (103, 'Aladár',    'Baba',     'muž',  '22.1.1980' ),
  (104, 'Ferdinand', 'Mravec',   'muž',  '3.3.1984'  ),
  (105, 'Ján',       'Polienko', 'muž',  '14.4.1982' ),
  (106, 'Juraj',     'Truľo',    'muž',  '16.7.1979' ),
  (107, 'Jana',      'Botková',  'žena', '21.9.1977' ),
  (108, 'Dana',      'Botková',  'žena', '21.9.1977' ),
  (109, 'Ján',       'Hlúpy',    'muž',  '1.4.1972'  ),
  (110, 'Aladár',    'Miazga',   'muž',  '22.12.1987'),
  (111, 'Mikuláš',   'Myšiak',   'muž',  '6.6.1983'  ),
  (112, 'Donald',    'Káčer',    'muž',  '7.10.1982' ),
  (113, 'Jozef',     'Námorník', 'muž',  '23.9.1981' ),
  (114, 'Peter',     'Pan',      'muž',  '13.1.2001' )
;
INSERT INTO učiteľ
VALUES
  (1, 1),
  (2, 1),
  (3, 2),
  (4, 2),
  (5, 3)
;
INSERT INTO študent
VALUES
  (101, 1, 1, 1,    NULL),
  (102, 1, 1, 1,    1   ),
  (103, 2, 2, 2,    4   ),
  (104, 2, 3, 3,    3   ),
```

```
  (105, 3, 5, 1,    3   ),
  (106, 3, 1, 1,    3   ),
  (107, 3, 4, NULL, 1   ),
  (108, 3, 4, NULL, 1   ),
  (109, 2, 2, 1,    2   ),
  (110, 2, 3, NULL, 4   ),
  (111, 1, 5, 4,    5   ),
  (112, 1, 5, 4,    5   ),
  (113, 1, 2, NULL, NULL),
  (114, 2, 1, 5,    NULL)
;
INSERT INTO člen_rady
VALUES
  (  1, 1, 2),
  (  2, 3, 1),
  (  4, 3, 1),
  (104, 3, 1)
;
INSERT INTO predmet
VALUES
  ( 1, 'UPR', 'Umenie pomáhať rozhovorom',                                  3, 4),
  ( 2, 'VZS', 'Sociálnopsychologický výcvik zvládania záťažových situácií', 2, 1),
  ( 3, 'NEM', 'Nebeská mechanika',                                          4, 1),
  ( 4, 'ZLO', 'Zložité systémy',                                            9, 5),
  ( 5, 'HNR', 'Hlbokonepružný rozptyl leptónov na hadrónoch',               8, 5),
  ( 6, 'MMO', 'Molekulový modeling',                                        3, 4),
  ( 7, 'ROM', 'Romológia',                                                  6, 3),
  ( 8, 'HRO', 'Teória hromadnej obsluhy',                                   4, 1),
  ( 9, 'HRY', 'Teória hier',                                                3, 3),
  (10, 'FYD', 'Fyzika DNA',                                                 3, 5),
  (11, 'FAK', 'Fázové prechody a kritické javy',                            5, 2),
  (12, 'DBS', 'Databázové systémy',                                         5, 2)
;
INSERT INTO prerekvizita
VALUES
  ( 3, 10),
  ( 6, 11),
  ( 7, 11),
  ( 8,  3),
  ( 8,  7),
  ( 9,  2),
  ( 9,  6),
  (10, 12),
  (11, 10)
;
INSERT INTO zapísané
VALUES
  (  1, 101,  1, '1.9.2003',   '20.5.2004',  2   ),       (  2, 101,  4, '1.9.2003',   NULL,         NULL),
  (  3, 101, 12, '1.9.2003',   NULL,         NULL),       (  4, 102,  1, '1.9.2003',   '15.5.2004',  3   ),
  (  5, 102,  4, '1.9.2003',   '20.6.2004',  3   ),       (  6, 102, 12, '1.9.2003',   NULL,         NULL),
  (  7, 106,  1, '1.9.2003',   '14.5.2004',  2   ),       (  8, 106,  4, '1.9.2003',   NULL,         NULL),
  (  9, 106, 12, '1.9.2003',   '5.2.2004',   1   ),       ( 10, 114,  1, '1.9.2003',   '8.5.2004',   1   ),
  ( 11, 114,  4, '1.9.2003',   '10.5.2004',  1   ),       ( 12, 114, 12, '1.9.2003',   '30.6.2004',  3   ),
  ( 13, 103,  1, '1.9.2002',   '22.5.2003',  2   ),       ( 14, 103,  4, '1.9.2002',   '27.5.2003',  3   ),
  ( 15, 103, 12, '1.9.2002',   '17.5.2003',  3   ),       ( 16, 103,  2, '1.9.2003',   NULL,         NULL),
  ( 17, 103,  5, '19.9.2003',  '12.5.2004',  1   ),       ( 18, 103, 10, '1.9.2003',   NULL,         NULL),
  ( 19, 109,  1, '1.9.2002',   '20.8.2003',  1   ),       ( 20, 109,  4, '1.9.2002',   '10.8.2003',  NULL),
  ( 21, 109,  4, '2.9.2003',   NULL,         NULL),       ( 22, 109, 12, '1.9.2003',   '24.5.2004',  2   ),
  ( 23, 109,  2, '1.9.2003',   '23.8.2004',  2   ),       ( 24, 109,  5, '1.9.2003',   '20.5.2004',  1   ),
  ( 25, 109, 10, '1.9.2003',   '10.5.2004',  2   ),       ( 26, 113,  1, '1.9.2002',   '14.6.2003',  2   ),
  ( 27, 113,  4, '1.9.2002',   NULL,         NULL),       ( 28, 113,  4, '11.10.2003', '20.7.2003',  3   ),
  ( 29, 113, 12, '1.9.2002',   '10.5.2003',  2   ),       ( 30, 113,  2, '2.9.2003',   '20.5.2004',  3   ),
  ( 31, 113,  5, '6.10.2003',  NULL,         NULL),       ( 32, 113,  6, '1.9.2003',   '20.5.2004',  3   ),
  ( 33, 113, 10, '1.9.2003',   NULL,         NULL),       ( 34, 104,  1, '1.9.2001',   '23.5.2002',  3   ),
  ( 35, 104,  4, '1.9.2001',   '20.5.2002',  2   ),       ( 36, 104, 12, '1.9.2001',   '21.5.2002',  1   ),
  ( 37, 104,  2, '3.9.2002',   '21.7.2003',  1   ),       ( 38, 104,  5, '1.9.2002',   '21.7.2003',  2   ),
  ( 39, 104,  6, '1.9.2002',   '12.5.2003',  3   ),       ( 40, 104, 10, '1.9.2003',   '22.5.2004',  3   ),
  ( 41, 104, 11, '1.9.2003',   '10.8.2004',  2   ),       ( 42, 104,  7, '1.9.2003',   '10.5.2004',  1   ),
  ( 43, 110,  1, '3.9.2001',   '7.8.2002',   3   ),       ( 44, 110,  4, '11.9.2001',  '20.5.2002',  2   ),
  ( 45, 110, 12, '1.9.2001',   NULL,         NULL),       ( 46, 110,  2, '1.9.2002',   '4.5.2003',   3   ),
  ( 47, 110,  5, '1.9.2002',   '6.6.2003',   3   ),       ( 48, 110,  6, '1.9.2002',   '16.5.2003',  2   ),
  ( 49, 110, 10, '5.9.2002',   '10.6.2003',  2   ),       ( 50, 110, 11, '3.9.2003',   '5.6.2004',   3   ),
  ( 51, 110,  8, '1.9.2003',   NULL,         NULL),       ( 52, 107,  1, '1.9.2000',   '2.8.2001',   1   ),
```

```
( 53, 107,  4, '1.9.2000',   '10.5.2001',  2   ),
( 55, 107,  2, '1.9.2001',   '20.8.2002',  1   ),
( 57, 107,  5, '2.9.2002',   '10.9.2003',  NULL),
( 59, 107,  6, '1.9.2002',   '24.5.2003',  3   ),
( 61, 107,  8, '1.9.2003',   NULL,         NULL),
( 63, 108,  4, '4.9.2000',   '25.5.2001',  1   ),
( 65, 108,  2, '1.9.2001',   '23.9.2002',  2   ),
( 67, 108,  5, '1.9.2002',   '17.5.2003',  1   ),
( 69, 108, 11, '1.9.2003',   NULL,         NULL),
( 71, 105,  4, '1.9.1999',   '4.5.2000',   1   ),
( 73, 105, 10, '1.9.2000',   '17.5.2001',  2   ),
( 75, 105,  3, '1.9.2001',   '15.8.2002',  3   ),
( 77, 105,  6, '2.9.2001',   '13.6.2002',  1   ),
( 79, 105,  8, '1.9.2002',   '22.5.2003',  2   ),
( 81, 105,  7, '1.9.2003',   NULL,         NULL),
( 83, 111,  4, '1.9.1999',   '10.8.2000',  1   ),
( 85, 111, 10, '1.9.2000',   '22.5.2001',  2   ),
( 87, 111,  3, '1.9.2001',   '23.5.2002',  2   ),
( 89, 111,  6, '1.9.2001',   '24.5.2002',  2   ),
( 91, 111,  8, '1.9.2002',   '23.8.2003',  2   ),
( 93, 111,  7, '1.9.2003',   NULL,         NULL),
( 95, 112,  4, '12.9.1999',  '19.5.2000',  3   ),
( 97, 112, 10, '1.9.2000',   '20.5.2001',  1   ),
( 99, 112,  3, '1.9.2001',   '10.5.2002',  2   ),
(101, 112,  6, '1.9.2002',   '20.8.2003',  3   ),
(103, 112,  8, '1.9.2002',   '24.10.2003', 2   ),
(105, 112,  7, '1.9.2003',   NULL,         NULL)

( 54, 107, 12, '1.9.2001',   '9.8.2002',   2   ),
( 56, 107,  3, '1.9.2001',   '15.9.2002',  2   ),
( 58, 107,  5, '11.9.2003',  NULL,         NULL),
( 60, 107, 11, '31.8.2003',  NULL,         NULL),
( 62, 108,  1, '1.9.2000',   '20.9.2001',  3   ),
( 64, 108, 12, '1.9.2000',   '24.6.2001',  1   ),
( 66, 108,  3, '1.9.2001',   '15.5.2002',  1   ),
( 68, 108,  6, '1.9.2002',   '5.7.2003',   2   ),
( 70, 105,  1, '11.9.1999',  '2.5.2000',   2   ),
( 72, 105, 12, '1.9.1999',   '15.5.2000',  1   ),
( 74, 105,  2, '6.9.2000',   '20.9.2001',  3   ),
( 76, 105,  5, '1.9.2001',   '20.5.2002',  2   ),
( 78, 105, 11, '3.9.2002',   '13.6.2003',  1   ),
( 80, 105,  9, '1.9.2003',   NULL,         NULL),
( 82, 111,  1, '1.9.1999',   '21.5.2000',  2   ),
( 84, 111, 12, '1.9.2000',   '11.5.2001',  2   ),
( 86, 111,  2, '1.9.2001',   '20.7.2002',  3   ),
( 88, 111,  5, '1.9.2001',   '13.8.2002',  2   ),
( 90, 111, 11, '1.9.2002',   '10.5.2003',  1   ),
( 92, 111,  9, '1.9.2002',   '23.6.2003',  2   ),
( 94, 112,  1, '1.9.1999',   '13.5.2000',  3   ),
( 96, 112, 12, '1.9.2000',   '15.5.2001',  2   ),
( 98, 112,  2, '11.11.2001', '23.5.2002',  1   ),
(100, 112,  5, '1.9.2001',   '21.5.2002',  1   ),
(102, 112, 11, '1.9.2002',   '12.5.2003',  3   ),
(104, 112,  9, '1.9.2003',   NULL,         NULL),
```

Obrázok databázového modelu teda mierne zmodifikujeme – entitné typy nahradíme menami zodpovedajúcich tabuliek a ich atribúty menami príslušných stĺpcov:



2.4.3 Revízia

Všimnime si dobre tabuľku `člen_rady`:

```
SELECT *
FROM člen_rady
```

Odpoveď:

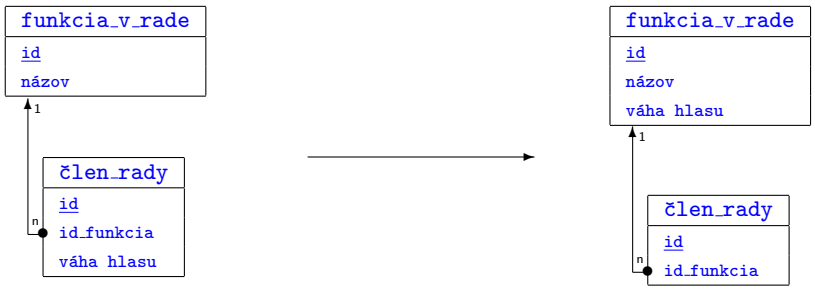
ID	ID_FUNKCIA	VÁHA_HLASU
1	1	2
2	3	1
4	3	1
104	3	1

Aj keď maličký počet záznamov môže skresľovať, zdá sa, že medzi hodnotami druhého a tretieho stĺpca existuje akási súvislosť, lebo každej hodnote stĺpca `id_funkcia` zodpovedá jedna hodnota stĺpca `váha_hlasu`

a naopak. Toto pozorovanie tak signalizuje možnú tesnejšiu väzbu medzi nimi. A naozaj, poctivejším nazretím do štatútu rady si overíme, že váha hlasu nie je závislá od toho-ktorého člena rady, ale od jeho funkcie v rade, je daná ex offo. Takýto vzťah, keď hodnoty stĺpca x jednoznačne určujú hodnoty stĺpca y, nazývame **funkčná závislosť** y od x (lebo y je v matematickom slova zmysle funkciou x). V našom prípade máme takýchto funkčných závislostí viac – keďže `id` je primárny kľúč, ostatné dva stĺpce sú od neho funkčne závislé, a to z definície. Platí to zrejme v každej tabuľke a pre každý (primárny či sekundárny) kľúč. Aj keď sa zdá, že naopak aj stĺpec `id_funkcia` je funkčne závislý od stĺpca `váha_hlasu`, ťažko tu hovoriť o ozajstnej funkčnej závislosti (stačí totiž zvoliť jedného člena na zatiaľ neobsadenú funkciu pokladníka, ktorý má podľa štatútu tiež jeden hlas ako ostatní členovia okrem predsedu, a zdanlivá jednoznačnosť je pasé).

Každá takto objavená funkčná závislosť má pre nás dva významy. V prvom rade nás upozorňuje na to, že pri návrhu databázy sme porušili pravidlo neduplicity dát, veď informáciu, že člen rady má jeden hlas, tu máme hneď trikrát (a informáciu o jednom hlase pokladníka vôbec nie). Last but not least **nás toto dodatočné poznanie vyvádza z pocitu neomylnosti, ukazuje nám, že ani pri najlepšej vôli nemožno očakávať, že databázový model sa podarí hneď na prvýkrát bez chyby**.

Ako teda svoj omyl napraviť? Stĺpec `id_funkcia` je cudzím kľúčom ukazujúcim na tabuľku `funkcia_v_rade`, preto funkčná závislosť stĺpca `váha_hlasu` od neho naznačuje, že tento stĺpec patrí do onej tabuľky. Napokon, znie to logicky, veď váha hlasu ktorémukoľvek členovi rady (včítane predsedu) prislúcha nie za jeho zásluhy, ale ex offo, z dôvodu jeho funkcie. Našťastie nemusíme prebudovávať hneď celú databázu, veď sa týka len tabuliek `člen_rady` a `funkcia_v_rade`:



Mohli by sme ich surovo vymazať a potom vytvoriť a naplniť nanovo, avšak pri tom, ako už sme viackrát upozorňovali, strácame dáta. Ďalšia možnosť je pridať stĺpec (do `funkcia_v_rade`), potrebujeme ho však aj mazať (z `člen_rady`), čo sa nedá. Urobíme to teda inak. V prvom rade sa (bez obáv zo zmiznutia dát) zbavíme všetkých cudzích kľúčov, ktoré sú na nich definované, resp. na ne ukazujú:

```
ALTER TABLE člen_rady DROP CONSTRAINT f_členrady_osoba
;
ALTER TABLE člen_rady DROP CONSTRAINT f_členrady_funkcia
```

Teraz tabuľky premenujeme pomocou príkazu `RENAME TABLE` („premenuj tabuľku“). Po tomto kľúčovom slove nasleduje starý názov tabuľky, za ním predložka `TO` (v tomto prípade „na“) a napokon nový (zatiaľ ešte inou tabuľkou neobsadený) názov:

```
RENAME TABLE člen_rady TO pom_člen_rady
;
RENAME TABLE funkcia_v_rade TO pom_funkcia_v_rade
```

Názvy sa uvoľnili, rýchlo ich využime na nové (dúfajme, správne) verzie našich tabuliek:

```
CREATE TABLE člen_rady
(
  id              INT         NOT NULL,
  id_funkcia      INT         NOT NULL,
  PRIMARY KEY (id)
)
;
CREATE TABLE funkcia_v_rade
(
  id              INT         NOT NULL,
  názov           VARCHAR(30) NOT NULL,
  váha_hlasu      INT         NOT NULL DEFAULT 1,
  PRIMARY KEY (id),
  CONSTRAINT u_funkcia_názov UNIQUE (názov)
)
```

Dodáme aj zmazané cudzie kľúče:

```
ALTER TABLE člen_rady ADD CONSTRAINT f_členrady_osoba FOREIGN KEY (id) REFERENCES osoba
  ON DELETE CASCADE
;
ALTER TABLE člen_rady ADD CONSTRAINT f_členrady_funkcia FOREIGN KEY (id_funkcia) REFERENCES funkcia_v_rade
```

A teraz ideme plniť. Vzhľadom na cudzí kľúč musíme začať tabuľkou `funkcia_v_rade`. Prvé dva stĺpce sú jednoduché, v treťom musíme použiť parametrický vnútorný dopyt (a v ňom, aby bol výsledok jednoznačný, použijeme trebárs `MAX`, ak sa hodnota nedá zo starých dát zistiť (ako napríklad u pokladníka), navolíme preddefinovanú `1`):

```
INSERT INTO funkcia_v_rade
SELECT
  id,
  názov,
  VALUE
  (
    (
      SELECT MAX(váha_hlasu)
      FROM pom_člen_rady AS pč
      WHERE pč.id_funkcia = pf.id
    ),
    1
  )
FROM pom_funkcia_v_rade AS pf
```

Naplniť druhú tabuľku nie je problém, tu nič dopĺňať nemusíme:

```
INSERT INTO člen_rady
SELECT
  id,
  id_funkcia
FROM pom_člen_rady
```

Overíme, že všetko vyšlo podľa našich predstáv:

```
SELECT *
FROM člen_rady


SELECT *
FROM funkcia_v_rade
```

Odpovede:

ID	ID_FUNKCIA
1	1
2	3
4	3
104	3

ID	NÁZOV	VÁHA_HLASU
1	predseda	2
2	pokladník	1
3	(radový) člen	1

Ostáva po týchto machináciách urobiť poriadok – odstrániť nepotrebné staré štruktúry (aj keď pod novými menami. . .):

```
DROP TABLE pom_člen_rady
;
DROP TABLE pom_funkcia_v_rade
```

2.5 Množinové operácie

2.5.1 Zjednotenie

Povedzme, že máme za úlohu vypísať usporiadaný zoznam priezvisk všetkých študentov i učiteľov. Nič jednoduchšie, veď sme potrebné údaje o nich prezieravo dali do jednej tabuľky:

```
SELECT priezvisko AS osoba
FROM osoba
ORDER BY 1
```

Odpoveď:

OSOBA
Baba
Baba
Botková
Botková
Hlúpy
Hraško
Káčer
Miazga
Miazga
Mravec
Múdry
Myšiak
Námorník
Pan
Polienko
Premúdra
Šípová
Truľo
Vševed

Ukázalo sa však, že takýto zoznam nepostačuje. Niektorí z učiteľov sa urazili, že nemajú uvedený titul, a pri študentoch zasa vyvstala požiadavka uviesť v zátvorke ich ročník so skratkou študijnej skupiny. Každú z týchto požiadaviek osobitne ľahko splníme. Najprv pri učiteľoch (vzhľadom na to, že každý učiteľ na tejto univerzite musí mať akademický titul, nemusíme použiť vonkajšie spojenie):

```
SELECT t.skratka || ' ' || o.priezvisko AS učiteľ
FROM
  osoba AS o
    JOIN učiteľ AS u ON o.id = u.id
      JOIN titul AS t ON u.id_titul = t.id
ORDER BY o.priezvisko
```

Odpoveď:

UČITEĽ
Mgr. Baba
.doc Miazga
.doc Múdry
Mgr. Premúdra
DEd Vševed

A teraz študenti (ani tu nie je potrebné vonkajšie spojenie):

```
SELECT o.priezvisko || ' ' || '(' || RTRIM(CHAR(š.ročník)) || RTRIM(s.kód) || ')' AS študent
FROM
  osoba AS o
    JOIN študent AS š ON o.id = š.id
      JOIN študijná_skupina AS s ON š.id_skupina = s.id
ORDER BY o.priezvisko
```

Odpoveď:

ŠTUDENT
Baba (2I)
Botková (4MI)
Botková (4MI)
Hlúpy (2I)
Hraško (1M)
Káčer (5M)
Miazga (3I)
Mravec (3I)
Myšiak (5M)
Námorník (2M)
Pan (1I)
Polienko (5MI)
Šípová (1M)
Truľo (1MI)

Ako tieto dve tabuľky zjednotiť do jednej? Odpoveď je skrytá v otázke – použijeme množinovú operáciu **zjednotenie**. Keď majú tabuľky rovnaký počet stĺpcov kompatibilného dátového typu (v našom prípade je to splnené – obe majú jediný stĺpec typu reťazec (na dĺžke nezáleží)), možno ich zjednotiť tak, že medzi zodpovedajúce dopyty (zbavené usporiadania) vložíme slová `UNION ALL` („zjednotenie všetkého“), a navyše obom výsledkom priradíme rovnaký alias:

```
SELECT t.skratka || ' ' || o.priezvisko AS osoba
FROM
  osoba AS o
    JOIN učiteľ AS u ON o.id = u.id
      JOIN titul AS t ON u.id_titul = t.id

UNION ALL

SELECT o.priezvisko || ' ' || '(' || RTRIM(CHAR(š.ročník)) || RTRIM(s.kód) || ')' AS osoba
FROM
  osoba AS o
    JOIN študent AS š ON o.id = š.id
      JOIN študijná_skupina AS s ON š.id_skupina = s.id
```

Odpoveď:

OSOBA
Hraško (1M)
Šípová (1M)
Myšiak (5M)
Káčer (5M)
Námorník (2M)
Baba (2I)
Mravec (3I)
Hlúpy (2I)
Miazga (3I)
Pan (1I)
Polienko (5MI)
Truľo (1MI)
Botková (4MI)
Botková (4MI)
.doc Miazga
.doc Múdry
Mgr. Premúdra
Mgr. Baba
DEd Vševed

Horšie to bude s usporiadaním podľa priezviska. Keďže zjednocujeme dve tabuľky, vo výsledkoch ktorých už nefiguruje stĺpec `priezvisko`, nemôžeme ho teda použiť v celkovom `ORDER BY`. Obyčajné usporiadanie síce áno, nedáva však to, čo potrebujeme:

```
SELECT t.skratka || ' ' || o.priezvisko AS osoba
FROM
  osoba AS o
    JOIN učiteľ AS u ON o.id = u.id
      JOIN titul AS t ON u.id_titul = t.id

UNION ALL

SELECT o.priezvisko || ' ' || '(' || RTRIM(CHAR(š.ročník)) || RTRIM(s.kód) || ')' AS osoba
FROM
  osoba AS o
    JOIN študent AS š ON o.id = š.id
      JOIN študijná_skupina AS s ON š.id_skupina = s.id

ORDER BY 1
```

Odpoveď:

OSOBA
.doc Miazga
.doc Múdry
Baba (2I)
Botková (4MI)
Botková (4MI)
DEd Vševed
Hlúpy (2I)
Hraško (1M)
Káčer (5M)
Mgr. Baba
Mgr. Premúdra
Miazga (3I)
Mravec (3I)
Myšiak (5M)
Námorník (2M)
Pan (1I)
Polienko (5MI)
Šípová (1M)
Truľo (1MI)

Použijeme preto vnorený dopyt, v ktorom stĺpec `priezvisko` bude:

```
SELECT osoba
FROM
(
  SELECT
    t.skratka || ' ' || o.priezvisko AS osoba,
    o.priezvisko AS priezvisko
  FROM
    osoba AS o
      JOIN učiteľ AS u ON o.id = u.id
        JOIN titul AS t ON u.id_titul = t.id

  UNION ALL

  SELECT
    o.priezvisko || ' ' || '(' || RTRIM(CHAR(š.ročník)) || RTRIM(s.kód) || ')' AS osoba,
    o.priezvisko AS priezvisko
  FROM
    osoba AS o
      JOIN študent AS š ON o.id = š.id
        JOIN študijná_skupina AS s ON š.id_skupina = s.id
) AS pomocná
ORDER BY priezvisko
```

Odpoveď:

OSOBA
Baba (2I)
Mgr. Baba
Botková (4MI)
Botková (4MI)
Hlúpy (2I)
Hraško (1M)
Káčer (5M)
Miazga (3I)
.doc Miazga
Mravec (3I)
.doc Múdry
Myšiak (5M)
Námorník (2M)
Pan (1I)
Polienko (5MI)
Mgr. Premúdra
Šípová (1M)
Truľo (1MI)
DEd Vševed

Tým sme úlohu splnili.

Aby sme v ďalšom situáciu sprehľadnili, pracujme opäť iba s priezviskami. Všimnime si, že v nasledujúcom príkaze má spojenie s tabuľkou učiteľ iba selektívnu funkciu – vypíšeme tak práve priezviská učiteľov:

```
SELECT o.priezvisko
FROM
  osoba AS o
    JOIN učiteľ AS u ON o.id = u.id
ORDER BY 1
```

Odpoveď:

PRIEZVISKO
Baba
Miazga
Múdry
Premúdra
Vševed

Analogicky sa môžeme obmedziť na študentov:

```
SELECT o.priezvisko
FROM
  osoba AS o
    JOIN študent AS š ON o.id = š.id
ORDER BY 1
```

Odpoveď:

PRIEZVISKO
Baba
Botková
Botková
Hlúpy
Hraško
Káčer
Miazga
Mravec
Myšiak
Námorník
Pan
Polienko
Šípová
Truľo

Zjednotením týchto dvoch dopytov zrejme dostaneme výpis priezvisk všetkých študentov a učiteľov (a keďže iné osoby ani neevidujeme, výsledok obsahuje priezviská všetkých zúčastnených):

```
SELECT o.priezvisko
FROM
  osoba AS o
    JOIN učiteľ AS u ON o.id = u.id

UNION ALL

SELECT o.priezvisko
FROM
  osoba AS o
    JOIN študent AS š ON o.id = š.id

ORDER BY 1
```

Odpoveď:

PRIEZVISKO
Baba
Baba
Botková
Botková
Hlúpy
Hraško
Káčer
Miazga
Miazga
Mravec
Múdry
Myšiak
Námorník
Pan
Polienko
Premúdra
Šípová
Truľo
Vševed

Ak však namiesto `UNION ALL` napíšeme len `UNION` („zjednotenie“), eliminujú sa duplicitné riadky, ako keby sme použili `DISTINCT` (a to dokonca aj v rámci oboch častí, čo vidieť na priezvisku `Botková`), a teda ide naozaj o množinové zjednotenie dvoch množín riadkov:

```
SELECT o.priezvisko
FROM
  osoba AS o
    JOIN učiteľ AS u ON o.id = u.id

UNION

SELECT o.priezvisko
FROM
  osoba AS o
    JOIN študent AS š ON o.id = š.id

ORDER BY 1
```

Odpoveď:

PRIEZVISKO
Baba
Botková
Hlúpy
Hraško
Káčer
Miazga
Mravec
Múdry
Myšiak
Námorník
Pan
Polienko
Premúdra
Šípová
Truľo
Vševed

2.5.2 Prienik a rozdiel

Okrem zjednotenia máme k dispozícii aj ďalšie množinové operácie. Priezviská, ktoré sa opakujú u študentov i učiteľov (podozrenie z protekcie?), dostaneme takto:

```
SELECT o.priezvisko
FROM
  osoba AS o
    JOIN učiteľ AS u ON o.id = u.id

INTERSECT

SELECT o.priezvisko
FROM
  osoba AS o
    JOIN študent AS š ON o.id = š.id

ORDER BY 1
```

Odpoveď:

PRIEZVISKO
Baba
Miazga

Syntax je rovnaká, použili sme však slovo INTERSECT („pretni“), ktoré zodpovedá množinovému **prieniku**. Poslednou, treťou operáciou, je množinový **rozdiel**, ktorý zapíšeme pomocou slova EXCEPT („okrem“):

```
SELECT o.priezvisko
FROM
  osoba AS o
    JOIN učiteľ AS u ON o.id = u.id

EXCEPT

SELECT o.priezvisko
FROM
  osoba AS o
    JOIN študent AS š ON o.id = š.id

ORDER BY 1
```

Odpoveď:

PRIEZVISKO
Múdry
Premúdra
Vševed

Dostali sme tak tie priezviská učiteľov, ktoré sa nevyskytujú medzi študentmi.

Obe tieto operácie majú tiež verziu s `ALL`.

2.5.3 Ich kombinácie

Všetky tri operácie možno použiť i na viac dopytov než na dva, dokonca ich možno navzájom kombinovať. Treba však pritom pamätať na podmienku kompatibility dátových typov stĺpcov jednotlivých častí, na nekomutativitu rozdielu a na prípadné zátvorkovanie. Najväčšiu prioritu má prienik, najmenšiu zhodne zjednotenie a rozdiel, rovnocenné časti sa vyhodnocujú v prirodzenom poradí zľava doprava (no, pri našom zápise skôr zhora nadol).

Napríklad symetrický rozdiel množín priezvisk učiteľov a študentov dostaneme takýmto dopytom:

```
(
  SELECT o.priezvisko
  FROM
    osoba AS o
      JOIN učiteľ AS u ON o.id = u.id

  EXCEPT

  SELECT o.priezvisko
  FROM
    osoba AS o
      JOIN študent AS š ON o.id = š.id
)
UNION
(
  SELECT o.priezvisko
  FROM
    osoba AS o
      JOIN študent AS š ON o.id = š.id

  EXCEPT

  SELECT o.priezvisko
  FROM
    osoba AS o
      JOIN učiteľ AS u ON o.id = u.id
)
ORDER BY 1
```

Odpoveď:

PRIEZVISKO
Botková
Hlúpy
Hraško
Káčer
Mravec
Múdry
Myšiak
Námorník
Pan
Polienko
Premúdra
Šípová
Truľo
Vševed

Bez zátvoriek však dostávame úplne iný výsledok:

```
SELECT o.priezvisko
FROM
```

```
  osoba AS o
    JOIN učiteľ AS u ON o.id = u.id

EXCEPT

SELECT o.priezvisko
FROM
  osoba AS o
    JOIN študent AS š ON o.id = š.id

UNION

SELECT o.priezvisko
FROM
  osoba AS o
    JOIN študent AS š ON o.id = š.id

EXCEPT

SELECT o.priezvisko
FROM
  osoba AS o
    JOIN učiteľ AS u ON o.id = u.id

ORDER BY 1
```

Odpoveď:

PRIEZVISKO
Botková
Hlúpy
Hraško
Káčer
Mravec
Myšiak
Námorník
Pan
Polienko
Šípová
Truľo

2.5.4 Kvantifikátory

Uvedomme si, že s množinami sme sa stretli už skôr – výsledok každého vnútorného dopytu sa dá považovať za množinu vyhovujúcich záznamov. Takýto vnútorný dopyt potom možno umiestniť do podmienky `IN` v klauzule `WHERE`. Napríklad ak chceme vypísať ľudí, ktorí sú členmi školskej rady, môžeme to urobiť aj takýmto spôsobom:

```
SELECT
  id,
  meno,
  priezvisko
FROM osoba
WHERE id IN (SELECT id FROM člen_rady)
```

Odpoveď:

ID	MENO	PRIEZVISKO
1	Gejza	Miazga
2	Matej	Múdry
4	Hedviga	Baba
104	Ferdinand	Mravec

Alternatívnym dopytom s rovnakým výsledkom je však aj toto:

```
SELECT
  id,
  meno,
  priezvisko
FROM osoba
WHERE id = SOME (SELECT id FROM člen_rady)
```

Použili sme tu slovo SOME („nejaký“), rovnosť v podmienke potom môžeme čítať, že identifikátor sa musí rovnať niektorému prvku množiny danej vnútorným dopytom. Ekvivalentnou náhradou za SOME je slovo ANY („dajaký“), takže náš dopyt môže vyzerať aj takto:

```
SELECT
  id,
  meno,
  priezvisko
FROM osoba
WHERE id = ANY (SELECT id FROM člen_rady)
```

SOME či ANY nemusíme použiť len pri rovnosti. Ak napríklad chceme zistiť, či nejaký študent nie je starší než niektorý učiteľ, môžeme napísať takýto dopyt:

```
SELECT
  o.id,
  meno,
  priezvisko
FROM
  osoba AS o
    JOIN študent AS š ON o.id = š.id
WHERE dátum_narodenia < SOME
      (
        SELECT dátum_narodenia
        FROM
          osoba AS o
            JOIN učiteľ AS u ON o.id = u.id
      )
```

Odpoveď:

ID	MENO	PRIEZVISKO
109	Ján	Hlúpy

Uvedomme si, že tieto funkcie zodpovedajú **existenčnému kvantifikátoru**. Druhý, **všeobecný kvantifikátor** možno vyjadriť slovom ALL („všetky“) na rovnakom mieste. Ak chceme napríklad nájsť učiteľov, ktorí sú starší než všetci študenti, napíšeme dopyt:

```
SELECT
  o.id,
  meno,
  priezvisko
FROM
  osoba AS o
    JOIN učiteľ AS u ON o.id = u.id
WHERE dátum_narodenia < ALL
      (
        SELECT dátum_narodenia
        FROM
          osoba AS o
            JOIN študent AS š ON o.id = š.id
      )
```

Odpoveď:

ID	MENO	PRIEZVISKO
1	Gejza	Miazga
2	Matej	Múdry
4	Hedviga	Baba
5	d’Eduard	Vševed

Existuje aj funkcia EXISTS („existuje“), ktorá odpovedá na otázku, či je množina v jej argumente neprázdna (a teda jej výsledok je pravdivostná hodnota). Ak chceme napríklad vypísať všetkých študentov, ktorí si zapísali HRO (predmet s číslom 8) alebo HRY (predmet s číslom 9), čiže takých, pre ktorých existuje príslušný záznam v tabuľke zapísané, môžeme napísať:

```
SELECT
  id,
  meno,
  priezvisko
FROM osoba AS o
WHERE EXISTS
      (
        SELECT *
        FROM
          zapísané AS z
            JOIN predmet p ON z.id_predmet = p.id
        WHERE
              z.id_študent = o.id
          AND p.kód IN ('HRO','HRY')
      )
```

Odpoveď:

ID	MENO	PRIEZVISKO
105	Ján	Polienko
107	Jana	Botková
110	Aladár	Miazga
111	Mikuláš	Myšiak
112	Donald	Káčer

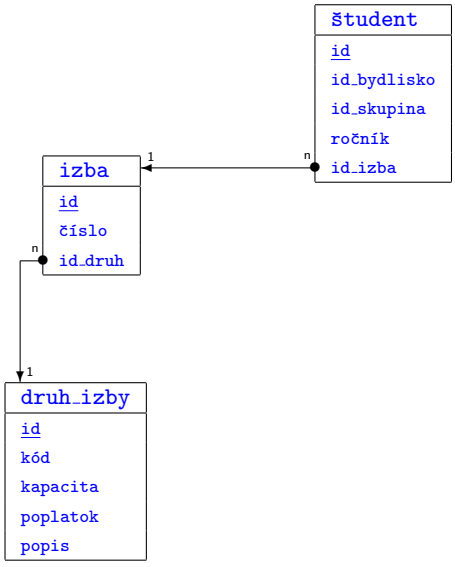
2.5.Ú Úlohy

1. Napíšte (bez použitia VALUES) dopyt, ktorého výsledok bude obsahovať meno a priezvisko každého učiteľa päťkrát za sebou, pričom týchto päť riadkov bude očíslovaných číslami 1 až 5.
2. Pri vytváraní našej databázy sa tabuľka krajina v istom prechodnom štádiu volala štát (prv než sme usúdili, že považovať Haligandu za štát by nebolo to pravé). Žiaľ, tento relikt sme zabudli vymazať, a on sa medzitým stal popri tabuľke krajina používanou časťou databázy. Navrhnite preto, ako obsah týchto dvoch tabuliek porovnať a prípadné rozdiely zahladiť.
3. Definujte situáciu, kde dávajú INTERSECT a INTERSECT ALL rozdielne výsledky.
4. Definujte situáciu, kde dávajú EXCEPT a EXCEPT ALL rozdielne výsledky.
5. Navrhnite, ako presvedčivo overiť prioritu zjednotenia, prieniku a rozdielu.

2.6 Ako postupovať pri konštrukcii dopytu?

2.6.1 Model ako grafická pomôcka

Vžime sa do úlohy riaditeľa internátu, ktorý chce zistiť celkovú sumu mesačných poplatkov od študentov. Relevantné informácie sa nachádzajú vo viacerých tabuľkách nášho systému – v tabuľke `druh_izby` sú uvedené poplatky, tabuľka `študent` obsahuje informáciu o tom, kde ten-ktorý študent býva, a tabuľka `izba` je ich prirodzeným sprostredkovateľom. (Uvedomme si tiež, že mená a priezviská študentov, a teda tabuľku `osoba`, nepotrebujeme.) Načrtnime si príslušnú časť nášho databázového modelu:



V prvej fáze vypíšme všetky ako-tak relevantné stĺpce zo všetkých troch tabuliek, všimnime si, že ich prepojenia v časti `FROM` sú dané cudzími kľúčmi medzi nimi:

```
SELECT
  š.id AS š_id,
  i.id AS i_id,
  d.id AS d_id,
  d.poplatok
FROM
  študent AS š
    JOIN izba AS i ON š.id_izba = i.id
      JOIN druh_izby AS d ON i.id_druh = d.id
```

Odpoveď:

Š_ID	I_ID	D_ID	POPLATOK
109	2	1	1000,00
103	4	4	600,00
110	4	4	600,00
111	5	4	600,00
112	5	4	600,00
102	1	6	400,00
107	1	6	400,00
108	1	6	400,00
104	3	6	400,00
105	3	6	400,00
106	3	6	400,00

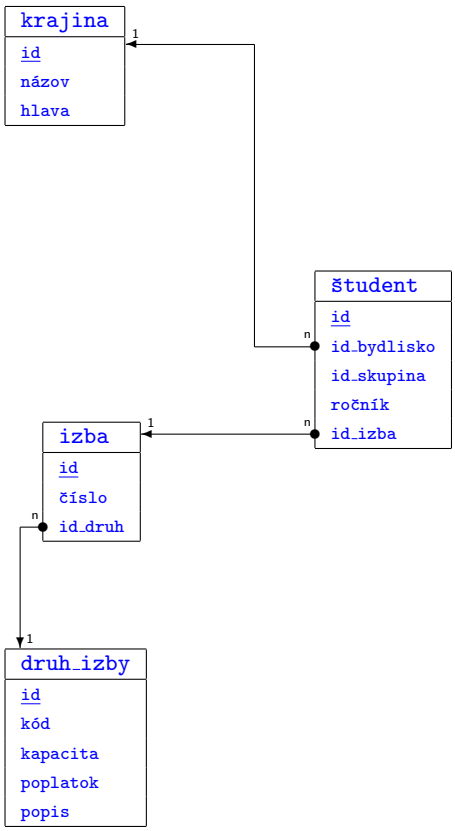
Všimnime si, že vo výsledku sú zahrnutí len tí študenti, ktorí bývajú na internáte, vylúčenie ostatných zariadilo napojenie tabuľky izba. Teraz už stačí iba sčítať poplatky, takže výsledok je:

```
SELECT SUM(d.poplatok) AS celkový_mesačný_poplatok
FROM
  študent AS š
    JOIN izba AS i ON š.id_izba = i.id
      JOIN druh_izby AS d ON i.id_druh = d.id
```

Odpoveď:

CELKOVÝ_MESAČNÝ_POPLATOK
5800,00

Riaditeľovu požiadavku sme tým vybavili, ten však medzitým zmenil názor – chce navyše rozdelenie tejto sumy na čiastky podľa trvalého bydliska študentov (možno chce vyberať poplatky v cudzej mene). K trojici tabuliek z predchádzajúcej úlohy teda treba pridať tabuľku krajina, kde sú uložené názvy krajín potrebné do výpisu. Tá je, ako ukazuje patričný výlomok z modelu, priamo napojená na tabuľku študent:



Najprv opäť výpis relevantných stĺpcov, pričom v časti FROM zasa rešpektujeme väzby z modelu:

```
SELECT
  š.id AS š_id,
  i.id AS i_id,
  d.id AS d_id,
  d.poplatok,
  k.id AS k_id,
  k.názov AS krajina
FROM
  študent AS š
    JOIN izba AS i ON š.id_izba = i.id
      JOIN druh_izby AS d ON i.id_druh = d.id
    JOIN krajina AS k ON š.id_bydlisko = k.id
```

Odpoveď:

Š_ID	I_ID	D_ID	POPLATOK	K_ID	KRAJINA
109	2	1	1000,00	1	Za siedmimi horami a siedmimi dolami
102	1	6	400,00	1	Za siedmimi horami a siedmimi dolami
105	3	6	400,00	1	Za siedmimi horami a siedmimi dolami
106	3	6	400,00	1	Za siedmimi horami a siedmimi dolami
103	4	4	600,00	2	Kalifát Bagdad
104	3	6	400,00	3	Mravenisko
111	5	4	600,00	4	Hollywood
112	5	4	600,00	4	Hollywood

Takže urobíme skupiny podľa názvu krajiny a pre každú zistíme celkový poplatok:

```
SELECT
  k.názov AS krajina,
  SUM(d.poplatok) AS celkový_mesačný_poplatok
FROM
  študent AS š
    JOIN izba AS i ON š.id_izba = i.id
      JOIN druh_izby AS d ON i.id_druh = d.id
    JOIN krajina AS k ON š.id_bydlisko = k.id
GROUP BY k.názov
```

Odpoveď:

KRAJINA	CELKOVÝ_MESAČNÝ_POPLATOK
Hollywood	1200,00
Kalifát Bagdad	600,00
Mravenisko	400,00
Za siedmimi horami a siedmimi dolami	2200,00

Všetko by to bolo pekné, keby sedela celková suma. Ale, žiaľ, nesedí – súčet poplatkov za jednotlivé krajiny je 4400,00, predtým nám však vyšlo 5800,00, čo je o 1400,00 viac. Kde je chyba? Mohlo nás to trknúť už skôr, keby sme si pri výpise všetkých relevantných stĺpcov všimli, že namiesto 11 riadkov je ich už len 8. Zvyšné tri zodpovedajú študentom s číslami 107, 108 a 110. Zistime, čo sú zač. Ich mená a priezviská nájdeme v tabuľke osoba, ostatné potrebné informácie sú v tabuľke študent. Na to, aby sme nezabudli na ich väzobnú podmienku, už hádam obrázok ani kresliť netreba:

```
SELECT
  o.meno,
  o.priezvisko,
  š.*
FROM
  študent AS š
    JOIN osoba AS o ON š.id = o.id
WHERE š.id IN (107, 108, 110)
```

Odpoveď:

MENO	PRIEZVISKO	ID	ID_SKUPINA	ROČNÍK	ID_BYDLISKO	ID_IZBA
Jana	Botková	107	3	4	NULL	1
Dana	Botková	108	3	4	NULL	1
Aladár	Miazga	110	2	3	NULL	4

Aha! Tak už vidíme, v čom je problém – títo traja študenti bývajú na internáte, ale nie je známe ich bydlisko. Preto sa ich neznáma krajina (alebo možno krajiny?) nedostala do zoznamu krajín, a celkový mesačný poplatok bol tým skreslený. Ľahko zistíme, že to sú práve tie chýbajúce peniaze:

```
SELECT
  š.id AS š_id,
  i.id AS i_id,
  d.id AS d_id,
  d.poplatok
FROM
  študent AS š
    JOIN izba AS i ON š.id_izba = i.id
      JOIN druh_izby AS d ON i.id_druh = d.id
WHERE š.id IN (107, 108, 110)
```

Odpoveď:

Š_ID	I_ID	D_ID	POPLATOK
110	4	4	600,00
107	1	6	400,00
108	1	6	400,00

alebo ešte lepšie:

```
SELECT SUM(d.poplatok) AS chýbajúce_prachy
FROM
  študent AS š
    JOIN izba AS i ON š.id_izba = i.id
      JOIN druh_izby AS d ON i.id_druh = d.id
WHERE š.id IN (107, 108, 110)
```

Odpoveď:

CHÝBAJÚCE_PRACHY
1400,00

Takže chybu (verme, že jedinú) sme našli, treba ju však odstrániť. Aby títo traja študenti ostali v zozname aj po napojení tabuľky `krajina`, musíme použiť vonkajšie spojenie. Takže:

```
SELECT
  k.názov AS krajina,
  SUM(d.poplatok) AS celkový_mesačný_poplatok
FROM
  študent AS š
    JOIN izba AS i ON š.id_izba = i.id
      JOIN druh_izby AS d ON i.id_druh = d.id
    LEFT OUTER JOIN krajina AS k ON š.id_bydlisko = k.id
GROUP BY k.názov
```

Odpoveď:

KRAJINA	CELKOVÝ_MESAČNÝ_POPLATOK
Hollywood	1200,00
Kalifát Bagdad	600,00
Mravenisko	400,00
Za siedmimi horami a siedmimi dolami	2200,00
NULL	1400,00

alebo s jemnou úpravou, aby sme sa vyhli prázdnej hodnote:

```
SELECT
  VALUE(k.názov,'ostatní') AS krajina,
  SUM(d.poplatok) AS celkový_mesačný_poplatok
FROM
  študent AS š
    JOIN izba AS i ON š.id_izba = i.id
      JOIN druh_izby AS d ON i.id_druh = d.id
    LEFT OUTER JOIN krajina AS k ON š.id_bydlisko = k.id
GROUP BY k.názov
ORDER BY k.názov
```

Odpoveď:

KRAJINA	CELKOVÝ_MESAČNÝ_POPLATOK
Hollywood	1200,00
Kalifát Bagdad	600,00
Mravenisko	400,00
Za siedmimi horami a siedmimi dolami	2200,00
ostatní	1400,00

Uvedomme si ešte, že pri ostatných tabuľkách je obyčajné (vnútorné) spojenie použité správne, ba dokonca v prípade tabuliek `študent` a `izba` žiaduce – obmedzíme ním množinu študentov len na tých, od ktorých poplatok môžeme reálne dostať. A riaditeľ internátu sa nám môže poďakovať za ďalšiu kladne vybavenú žiadosť.

2.6.2 Pomocné tabuľky

Ďalším riaditeľovým nápadom nech je kontrola prekročenia kapacity izieb. Výsledkom dopytu by mal byť zoznam izieb, kde býva viac študentov, než predpisy dovoľujú. Pre každú izbu vieme ľahko zistiť jej kapacitu – stačí na tabuľku `izba` napojiť tabuľku `druh_izby` (ak treba, môžeme si, samozrejme, aj tu pomôcť načrtnutím relevantnej časti modelu):

```
SELECT
  i.číslo AS izba,
  d.kapacita
FROM
  izba AS i
    JOIN druh_izby AS d ON i.id_druh = d.id
```

Odpoveď:

IZBA	KAPACITA
013A	1
323A	1
354B	2
321A	2
101A	3
242B	3

Zistiť reálne obsadenie izieb tiež nie je náročné – na tabuľku `izba` napojíme tabuľku `študent` a záznamy zoskupinujeme práve podľa čísla izby:

```
SELECT
  i.číslo AS izba,
  COUNT(š.id) AS obsadenie
FROM
  izba AS i
    JOIN študent AS š ON š.id_izba = i.id
GROUP BY i.číslo
```

Odpoveď:

IZBA	OBSADENIE
013A	1
101A	3
242B	3
321A	2
354B	2

Takto sme však zistili len obsadenie neprázdnych izieb. Ak chceme v zozname aj neobsadené, použijeme vonkajšie spojenie:

```
SELECT
  i.číslo AS izba,
  COUNT(š.id) AS obsadenie
FROM
  izba AS i
    LEFT OUTER JOIN študent AS š ON š.id_izba = i.id
GROUP BY i.číslo
```

Odpoveď:

IZBA	OBSADENIE
013A	1
101A	3
242B	3
321A	2
323A	0
354B	2

Máme teda dve tabuľky, otázkou je, ako ich spojiť do jednej a výsledky porovnať. Dobre nám tu poslúžia **pomocné tabuľky**. Dopyt sa potom začína nie slovom SELECT, ale WITH („s“). Za ním nasledujú definície jednej alebo viacerých pomocných tabuliek oddelené čiarkami. Každá sa začína názvom pomocnej tabuľky a zátvorkami so zoznamom názvov jej stĺpcov. Potom ide (tentoraz povinné) slovo AS a za ním definícia vytvorenia jej dát (t. j. vnorený dopyt), pričom uvedený počet stĺpcov musí korešpondovať s definovaným počtom stĺpcov (dátové typy novej tabuľky sa neuvádzajú, sú dané implicitne definíciou dát). V takejto definícii sa môžu používať ako štandardné tabuľky, tak už definované tabuľky pomocné (dokonca, ako uvidíme v podkapitole o tranzitívnom uzávere, povolená je aj rekurzia). Ak sa navrhované názvy stĺpcov (po názve pomocnej tabuľky) zhodujú s názvami stĺpcov definovanými vo vnútornom dopyte, zoznam stĺpcov možno vynechať. Po definícii pomocnej tabuľky už nasleduje klasický dopyt, v ktorom môžeme okrem klasických tabuliek používať aj práve definované pomocné.

Takže môžeme písať:

```
WITH
  kapacita (izba, kapacita) AS
  (
    SELECT
      i.číslo AS izba,
      d.kapacita
    FROM
      izba AS i
        JOIN druh_izby AS d ON i.id_druh = d.id
  ),
```

```
  obsadenie (izba, obsadenie) AS
  (
    SELECT
      i.číslo AS izba,
      COUNT(š.id) AS obsadenie
    FROM
      izba AS i
        LEFT OUTER JOIN študent AS š ON š.id_izba = i.id
    GROUP BY i.číslo
  )
SELECT
  k.izba,
  k.kapacita,
  o.obsadenie
FROM
  kapacita k
    JOIN obsadenie o ON k.izba = o.izba
```

alebo (ekvivalentne) bez uvedeného zoznamu stĺpcov:

```
WITH
  kapacita AS
  (
    SELECT
      i.číslo AS izba,
      d.kapacita
    FROM
      izba AS i
        JOIN druh_izby AS d ON i.id_druh = d.id
  ),
  obsadenie AS
  (
    SELECT
      i.číslo AS izba,
      COUNT(š.id) AS obsadenie
    FROM
      izba AS i
        LEFT OUTER JOIN študent AS š ON š.id_izba = i.id
    GROUP BY i.číslo
  )
SELECT
  k.izba,
  k.kapacita,
  o.obsadenie
FROM
  kapacita k
    JOIN obsadenie o ON k.izba = o.izba
```

Odpoveď:

IZBA	KAPACITA	OBSADENIE
013A	1	1
101A	3	3
242B	3	3
321A	2	2
323A	1	0
354B	2	2

Dodajme, že korektnejšie by bolo v oboch pomocných tabuľkách namiesto stĺpca `izba.číslo` pracovať s primárnym kľúčom `izba.id`, avšak vzhľadom na to, že aj stĺpec `izba.číslo` je už zo svojej definície jednoznačný (je naň položená podmienka `UNIQUE`), mohli sme si dovoliť toto zjednodušenie. Teraz už stačí vypísať iba izby, kde obsadenie presahuje kapacitu:

```
WITH
  kapacita AS
  (
    SELECT
      i.číslo AS izba,
      d.kapacita
    FROM
      izba AS i
        JOIN druh_izby AS d ON i.id_druh = d.id
  ),
  obsadenie AS
  (
    SELECT
      i.číslo AS izba,
      COUNT(š.id) AS obsadenie
    FROM
      izba AS i
        LEFT OUTER JOIN študent AS š ON š.id_izba = i.id
    GROUP BY i.číslo
  )
SELECT k.izba
FROM
  kapacita k
    JOIN obsadenie o ON k.izba = o.izba
WHERE k.kapacita < o.obsadenie
```

Odpoveď:

IZBA

Vidíme teda, že obsadenie všetkých izieb je korektné. Aby sme sa o správnosti nášho dopytu uistili, presťahujme na chvíľu trebárs Jána Hlúpeho (s číslom 109) z izby 013A (s číslom 2) do izby 321A (s číslom 5), čím presiahneme jej kapacitu:

```
UPDATE študent
SET id_izba = 5
WHERE id = 109
```

Teraz bude výsledok predchádzajúceho dopytu takýto:

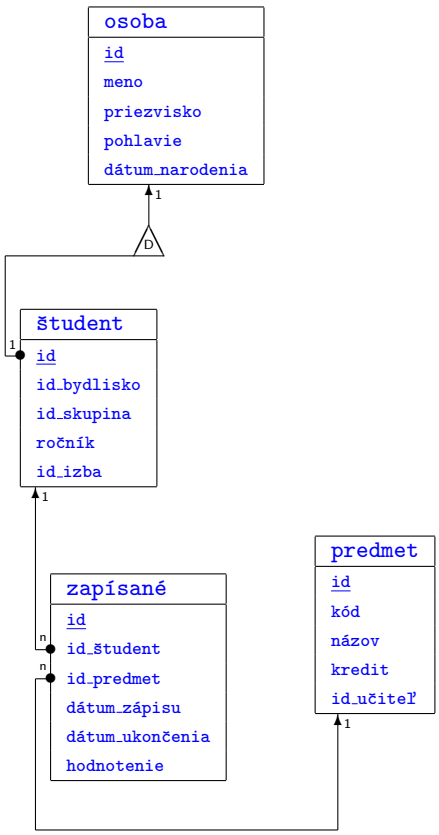
IZBA
321A

Aby Hlúpy nemal problémy aspoň na internáte, presťahujme ho rýchlo späť:

```
UPDATE študent
SET id_izba = 2
WHERE id = 109
```

2.6.3 Pomocné tabuľky a ich pomocné tabuľky

Metódu pomocných tabuliek môžeme použiť aj na výpis prehľadu možných a získaných kreditov (a ich pomeru) jednotlivých študentov. Počet možných kreditov každého študenta je vlastne súčet počtov kreditov ním zapísaných predmetov (každého však iba raz). Potrebujeme teda na centrálnu tabuľku zapísané, kde je drvivá väčšina relevantných údajov, napojiť jednak tabuľku predmet, kde je informácia o kreditoch, jednak tabuľku osoba, kde sú mená a priezviská študentov. Keďže však tabuľky osoba a zapísané nie sú prepojené priamo, svoje sprostredkovateľské služby ponúka tabuľka študent. Radšej načrtnime:



Opäť vypíšeme ako-tak dôležité údaje. Musíme však pamätať aj na študentov, ktorí si prípadne ešte zatiaľ nezapísali nič, a medzi tabuľkami `študent` a `zapísané` použiť vonkajšie spojenie. Takže predbežne (kvôli prehľadnosti) vypíšme iba údaje o Jane Botkovej s číslom 107. Všimnime si pri tom zátvorkovanie vo `FROM`:

```
SELECT
  o.id AS o_id,
  o.meno,
  o.priezvisko,
  p.id AS p_id,
  p.názov AS predmet,
  p.kredit
FROM
  zapísané AS z
    JOIN predmet AS p ON z.id_predmet = p.id
    RIGHT OUTER JOIN
      (
        študent AS š
          JOIN osoba AS o ON o.id = š.id
      ) ON z.id_študent = š.id
WHERE o.id = 107
ORDER BY p_id
```

Odpoveď:

O_ID	MENO	PRIEZVISKO	P_ID	PREDMET	KREDIT
107	Jana	Botková	1	Umenie pomáhať rozhovorom	3
107	Jana	Botková	2	Sociálnopsychologický výcvik zvládania záťažových situácií	2
107	Jana	Botková	3	Nebeská mechanika	4
107	Jana	Botková	4	Zložité systémy	9
107	Jana	Botková	5	Hlbokonepružný rozptyl leptónov na hadrónoch	8
107	Jana	Botková	5	Hlbokonepružný rozptyl leptónov na hadrónoch	8
107	Jana	Botková	6	Molekulový modeling	3
107	Jana	Botková	8	Teória hromadnej obsluhy	4
107	Jana	Botková	11	Fázové prechody a kritické javy	5
107	Jana	Botková	12	Databázové systémy	5

Vidíme, že Jana Botková si zapísala Hlbokonepružný rozptyl leptónov na hadrónoch dvakrát (zrejme prvý raz neuspela), do celkového počtu možných kreditov ho však môžeme započítať iba raz. Urobíme si teda najprv pomocnú tabuľku, kde vypíšeme každý zapísaný predmet iba raz, a súčet urobíme v ďalšom dopyte. Teraz to už urobme pre všetkých študentov:

```
WITH zapísaný_predmet (id_študent, meno, priezvisko, id_predmet, kredit) AS
(
  SELECT DISTINCT
    o.id,
    o.meno,
    o.priezvisko,
    p.id,
    p.kredit
  FROM
    zapísané AS z
      JOIN predmet AS p ON z.id_predmet = p.id
      RIGHT OUTER JOIN
        (
          študent AS š
            JOIN osoba AS o ON o.id = š.id
        ) ON z.id_študent = š.id
)
SELECT
  id_študent AS id,
  meno,
  priezvisko,
  SUM(kredit) AS možné_kredity
FROM zapísaný_predmet
GROUP BY
  id_študent,
  meno,
  priezvisko
```

Odpoveď:

ID	MENO	PRIEZVISKO	MOŽNÉ_KREDITY
101	Ján	Hraško	17
102	Ružena	Šípová	17
103	Aladár	Baba	30
104	Ferdinand	Mravec	44
105	Ján	Polienko	55
106	Juraj	Truľo	17
107	Jana	Botková	43
108	Dana	Botková	39
109	Ján	Hlúpy	30
110	Aladár	Miazga	42
111	Mikuláš	Myšiak	55
112	Donald	Káčer	55
113	Jozef	Námorník	33
114	Peter	Pan	17

Situácia pri získaných kreditoch bude úplne analogická, navyše však položíme podmienku, že študent získal za príslušný predmet nejaké hodnotenie:

```
WITH absolvovaný_predmet (id_študent, meno, priezvisko, id_predmet, kredit) AS
(
  SELECT DISTINCT
    o.id,
    o.meno,
    o.priezvisko,
    p.id,
    p.kredit
  FROM
    zapísané AS z
      JOIN predmet AS p ON z.id_predmet = p.id
      RIGHT OUTER JOIN
        (
          študent AS š
            JOIN osoba AS o ON o.id = š.id
        ) ON z.id_študent = š.id
  WHERE z.hodnotenie IS NOT NULL
)
SELECT
  id_študent AS id,
  meno,
  priezvisko,
  SUM(kredit) AS získané_kredity
FROM absolvovaný_predmet
GROUP BY
  id_študent,
  meno,
  priezvisko
```

Odpoveď:

ID	MENO	PRIEZVISKO	ZÍSKANÉ_KREDITY
101	Ján	Hraško	3
102	Ružena	Šípová	12
103	Aladár	Baba	25
104	Ferdinand	Mravec	44
105	Ján	Polienko	46
106	Juraj	Truľo	8
107	Jana	Botková	26
108	Dana	Botková	34
109	Ján	Hlúpy	21
110	Aladár	Miazga	33
111	Mikuláš	Myšiak	49
112	Donald	Káčer	46
113	Jozef	Námorník	22
114	Peter	Pan	17

Teraz ešte treba tieto dva medzivýsledky spojiť, čo urobíme tak ako minule. Všimnime si, že tentoraz máme štyri pomocné tabuľky, z ktorých druhá používa prvú a štvrtá tretiu – áno, ako sme už povedali, pomocné tabuľky sa môžu medzi sebou volať, volajúca však musí byť definovaná až po volanej:

```
WITH
  zapísaný_predmet (id_študent, meno, priezvisko, id_predmet, kredit) AS
  (
    SELECT DISTINCT
      o.id,
      o.meno,
      o.priezvisko,
      p.id,
      p.kredit
    FROM
      zapísané AS z
        JOIN predmet AS p ON z.id_predmet = p.id
        RIGHT OUTER JOIN
          (
            študent AS š
            JOIN osoba AS o ON o.id = š.id
          ) ON z.id_študent = š.id
  ),
```

```
  možné_kredity (id, meno, priezvisko, možné_kredity) AS
  (
    SELECT
      id_študent,
      meno,
      priezvisko,
      SUM(kredit)
    FROM zapísaný_predmet
    GROUP BY
      id_študent,
      meno,
      priezvisko
  ),
  absolvovaný_predmet (id_študent, meno, priezvisko, id_predmet, kredit) AS
  (
    SELECT DISTINCT
      o.id,
      o.meno,
      o.priezvisko,
      p.id,
      p.kredit
    FROM
      zapísané AS z
        JOIN predmet AS p ON z.id_predmet = p.id
        RIGHT OUTER JOIN
          (
            študent AS š
              JOIN osoba AS o ON o.id = š.id
          ) ON z.id_študent = š.id
    WHERE z.hodnotenie IS NOT NULL
  ),
  získané_kredity (id, meno, priezvisko, získané_kredity) AS
  (
    SELECT
      id_študent,
      meno,
      priezvisko, SUM(kredit)
    FROM absolvovaný_predmet
    GROUP BY
      id_študent,
      meno,
      priezvisko
  )
SELECT
  m.id,
  m.meno,
  m.priezvisko,
  m.možné_kredity,
  z.získané_kredity
FROM
  možné_kredity m
    JOIN získané_kredity z ON m.id = z.id
```

Odpoveď:

ID	MENO	PRIEZVISKO	MOŽNÉ_KREDITY	ZÍSKANÉ_KREDITY
101	Ján	Hraško	17	3
102	Ružena	Šípová	17	12
103	Aladár	Baba	30	25
104	Ferdinand	Mravec	44	44
105	Ján	Polienko	55	46
106	Juraj	Truľo	17	8
107	Jana	Botková	43	26
108	Dana	Botková	39	34
109	Ján	Hlúpy	30	21
110	Aladár	Miazga	42	33
111	Mikuláš	Myšiak	55	49
112	Donald	Káčer	55	46
113	Jozef	Námorník	33	22
114	Peter	Pan	17	17

Všimnime si, že stĺpce meno a priezvisko sa v pomocných tabuľkách vyskytujú zbytočne často. Odkiaľ ich vyhodiť? Odstránime ich pre istotu zovšadiaľ, a napojíme ich potom až v hlavnom dopyte. Odpoveď na tento dopyt je totožná s predchádzajúcou:

```
WITH
  zapísaný_predmet (id_študent, id_predmet, kredit) AS
  (
    SELECT DISTINCT
      o.id,
      p.id,
      p.kredit
    FROM
      zapísané AS z
        JOIN predmet AS p ON z.id_predmet = p.id
        RIGHT OUTER JOIN
          (
            študent AS š
            JOIN osoba AS o ON o.id = š.id
          ) ON z.id_študent = š.id
  ),
  možné_kredity (id, možné_kredity) AS
  (
    SELECT
      id_študent,
      SUM(kredit)
    FROM zapísaný_predmet
    GROUP BY id_študent
  ),
  absolvovaný_predmet (id_študent, id_predmet, kredit) AS
  (
    SELECT DISTINCT
      o.id,
      p.id,
      p.kredit
    FROM
      zapísané AS z
        JOIN predmet AS p ON z.id_predmet = p.id
        RIGHT OUTER JOIN
          (
            študent AS š
              JOIN osoba AS o ON o.id = š.id
          ) ON z.id_študent = š.id
    WHERE z.hodnotenie IS NOT NULL
  ),
  získané_kredity (id, získané_kredity) AS
  (
    SELECT
      id_študent,
      SUM(kredit)
    FROM absolvovaný_predmet
    GROUP BY id_študent
  )
SELECT
  o.id,
  o.meno,
  o.priezvisko,
  VALUE(m.možné_kredity,0),
  VALUE(z.získané_kredity,0)
FROM
  osoba AS o
    JOIN študent AS š ON o.id = š.id
    JOIN možné_kredity AS m ON š.id = m.id
    JOIN získané_kredity AS z ON š.id = z.id
```

Za zmienku stojí, že tabuľku študent by sme z hlavného dopytu mohli vyhodiť, už tabuľka možné_kredity (ale tiež tabuľka získané_kredity) zabezpečí, že vo výsledku iné osoby ako študenti figurovať nebudú. Pravdaže, v takom prípade treba š.id v posledných dvoch riadkoch nahradiť o.id; využívame tu, že medzi tabuľkami študent a osoba je vzťah 1:1, teda primárny kľúč študent.id je zároveň cudzím kľúčom do tabuľky osoba. Prostredníctvo tabuľky študent tak nie je nevyhnutné.

2.6.4 Viac exemplárov tej istej tabuľky

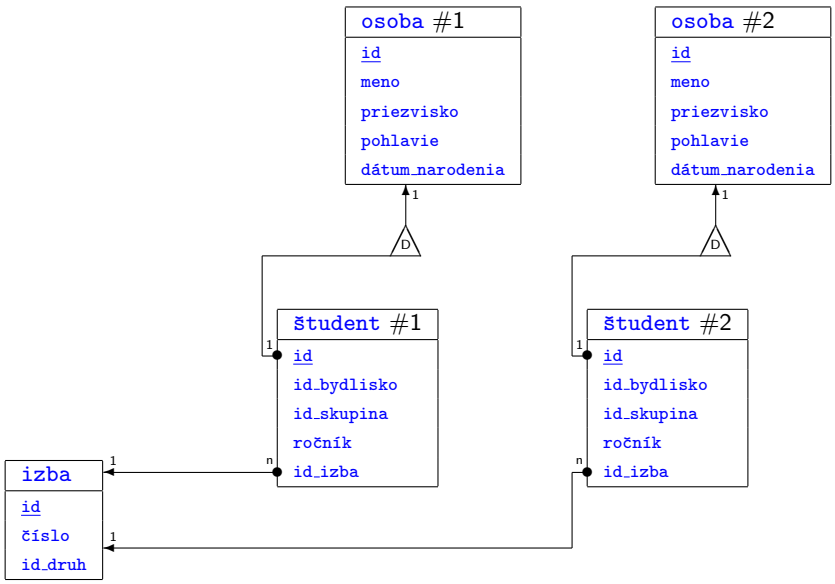
A opäť úloha z internátu – treba vypísať všetky dvojice študentov bývajúcich na jednej izbe (včítane čísla izby). Skúsme najprv vypísať pre každého študenta číslo jeho izby. Vieme, že ich mená a priezviská sú uložené v tabuľke `osoba`, v tabuľke `študent` sa skrýva informácia o tom, kde ktorý študent býva, a číslo izby je v tabuľke `izba`. Takže dopyt vyzerá takto:

```
SELECT
  o.id,
  o.meno,
  o.priezvisko,
  i.číslo AS izba
FROM
  izba AS i
    JOIN študent AS š ON š.id_izba = i.id
      JOIN osoba AS o ON o.id = š.id
```

Odpoveď:

ID	MENO	PRIEZVISKO	IZBA
102	Ružena	Šípová	101A
107	Jana	Botková	101A
108	Dana	Botková	101A
109	Ján	Hlúpy	013A
104	Ferdinand	Mravec	242B
105	Ján	Polienko	242B
106	Juraj	Truľo	242B
103	Aladár	Baba	354B
110	Aladár	Miazga	354B
111	Mikuláš	Myšiak	321A
112	Donald	Káčer	321A

Vidíme, že napríklad Mikuláš Myšiak a Donald Káčer bývajú v tej istej izbe, a preto by sa obe ich mená mali vyskytnúť v tom istom riadku výsledku. Ako to však docieliť? Vezmime si ľubovoľného študenta, k nemu hľadáme všetkých jeho spolubývajúcich. Ich mená sa určite nachádzajú v tabuľke `osoba`. Pre pevný riadok (v tabuľke `osoba`) prislúchajúci tomuto študentovi teda hľadáme možno niekoľko riadkov tabuľky `osoba`. Na jednej strane teda potrebujeme tabuľku `osoba` fixovať na nejakom riadku, na strane druhej ju chceme prehľadávať. Tento paradox nás prirodzene vedie k myšlienke použiť v dopyte dva rôzne exemplár tejto tabuľky. Aby však pre nejednoznačnosť názvu dvakrát použitej tabuľky `osoba` nezlyhal, musíme oba exemplár odlíšiť aliasmi (pravdaže, stačil by jeden, ale **zmysel pre symetriu** to nedovolí). Situáciu teda môžeme znázorniť nasledujúcim obrázkom:



Skúsme teda zodpovedajúci dopyt:

```
SELECT
  o1.id AS id1,
  o1.meno AS meno1,
  o1.priezvisko AS priezvisko1,
  o2.id AS id2,
  o2.meno AS meno2,
  o2.priezvisko AS priezvisko2,
  i.číslo AS izba
FROM
  izba AS i
    JOIN študent AS š1 ON š1.id_izba = i.id
      JOIN osoba AS o1 ON o1.id = š1.id
    JOIN študent AS š2 ON š2.id_izba = i.id
      JOIN osoba AS o2 ON o2.id = š2.id
```

Odpoveď:

ID1	MENO1	PRIEZVISKO1	ID2	MENO2	PRIEZVISKO2	IZBA
102	Ružena	Šípová	102	Ružena	Šípová	101A
107	Jana	Botková	102	Ružena	Šípová	101A
108	Dana	Botková	102	Ružena	Šípová	101A
102	Ružena	Šípová	107	Jana	Botková	101A
107	Jana	Botková	107	Jana	Botková	101A
108	Dana	Botková	107	Jana	Botková	101A
102	Ružena	Šípová	108	Dana	Botková	101A
107	Jana	Botková	108	Dana	Botková	101A
108	Dana	Botková	108	Dana	Botková	101A
109	Ján	Hlúpy	109	Ján	Hlúpy	013A
104	Ferdinand	Mravec	104	Ferdinand	Mravec	242B
105	Ján	Polienko	104	Ferdinand	Mravec	242B
106	Juraj	Truľo	104	Ferdinand	Mravec	242B
104	Ferdinand	Mravec	105	Ján	Polienko	242B
105	Ján	Polienko	105	Ján	Polienko	242B
106	Juraj	Truľo	105	Ján	Polienko	242B
104	Ferdinand	Mravec	106	Juraj	Truľo	242B
105	Ján	Polienko	106	Juraj	Truľo	242B
106	Juraj	Truľo	106	Juraj	Truľo	242B
103	Aladár	Baba	103	Aladár	Baba	354B
110	Aladár	Miazga	103	Aladár	Baba	354B
103	Aladár	Baba	110	Aladár	Miazga	354B
110	Aladár	Miazga	110	Aladár	Miazga	354B
111	Mikuláš	Myšiak	111	Mikuláš	Myšiak	321A
112	Donald	Káčer	111	Mikuláš	Myšiak	321A
111	Mikuláš	Myšiak	112	Donald	Káčer	321A
112	Donald	Káčer	112	Donald	Káčer	321A

No, skoro dobre, až na to, že v niektorých riadkoch je tá istá osoba dvakrát. Ľahko to napravíme dodatočnou podmienkou na nerovnosť identifikátorov:

```
SELECT
  o1.id AS id1,
  o1.meno AS meno1,
  o1.priezvisko AS priezvisko1,
  o2.id AS id2,
  o2.meno AS meno2,
  o2.priezvisko AS priezvisko2,
  i.číslo AS izba
FROM
  izba AS i
    JOIN študent AS š1 ON š1.id_izba = i.id
      JOIN osoba AS o1 ON o1.id = š1.id
    JOIN študent AS š2 ON š2.id_izba = i.id
      JOIN osoba AS o2 ON o2.id = š2.id
WHERE o1.id <> o2.id
```

Odpoveď:

ID1	MENO1	PRIEZVISKO1	ID2	MENO2	PRIEZVISKO2	IZBA
107	Jana	Botková	102	Ružena	Šípová	101A
108	Dana	Botková	102	Ružena	Šípová	101A
102	Ružena	Šípová	107	Jana	Botková	101A
108	Dana	Botková	107	Jana	Botková	101A
102	Ružena	Šípová	108	Dana	Botková	101A
107	Jana	Botková	108	Dana	Botková	101A
105	Ján	Polienko	104	Ferdinand	Mravec	242B
106	Juraj	Truľo	104	Ferdinand	Mravec	242B
104	Ferdinand	Mravec	105	Ján	Polienko	242B
106	Juraj	Truľo	105	Ján	Polienko	242B
104	Ferdinand	Mravec	106	Juraj	Truľo	242B
105	Ján	Polienko	106	Juraj	Truľo	242B
110	Aladár	Miazga	110	Aladár	Baba	354B
103	Aladár	Baba	110	Aladár	Miazga	354B
112	Donald	Káčer	111	Mikuláš	Myšiak	321A
111	Mikuláš	Myšiak	112	Donald	Káčer	321A

Takisto si všimnime, že každá už ozajstná dvojica je v dvoch riadkoch – raz tak a raz v opačnom poradí. Je otázne, ako to zadávateľ myslel, ale ak by to malo prekážať, jednoducho povieme, že ľudia v dvojici budú nejako usporiadaní, najlepšie podľa abecedy (a v prípade totožnosti mien a priezvisk (opäť si spomeňme na Jánov Hlúpych...) podľa identifikátorov). Potom je už podmienka na rôznosť identifikátorov nadbytočná. Takže upravený dopyt vyzerá takto:

```
SELECT
  o1.id AS id1,
  o1.meno AS meno1,
  o1.priezvisko AS priezvisko1,
  o2.id AS id2,
  o2.meno AS meno2,
  o2.priezvisko AS priezvisko2,
  i.číslo AS izba
FROM
  izba AS i
    JOIN študent AS š1 ON š1.id_izba = i.id
      JOIN osoba AS o1 ON o1.id = š1.id
    JOIN študent AS š2 ON š2.id_izba = i.id
      JOIN osoba AS o2 ON o2.id = š2.id
WHERE
      (o1.priezvisko < o2.priezvisko)
   OR (o1.priezvisko = o2.priezvisko AND o1.meno < o2.meno)
   OR (o1.priezvisko = o2.priezvisko AND o1.meno = o2.meno AND o1.id < o2.id)
ORDER BY 3, 2, 1, 6, 5, 4
```

Odpoveď:

ID1	MENO1	PRIEZVISKO1	ID2	MENO2	PRIEZVISKO2	IZBA
103	Aladár	Baba	110	Aladár	Miazga	354B
108	Dana	Botková	107	Jana	Botková	101A
108	Dana	Botková	102	Ružena	Šípová	101A
107	Jana	Botková	102	Ružena	Šípová	101A
112	Donald	Káčer	111	Mikuláš	Myšiak	321A
104	Ferdinand	Mravec	105	Ján	Polienko	242B
104	Ferdinand	Mravec	106	Juraj	Truľo	242B
105	Ján	Polienko	106	Juraj	Truľo	242B

Teraz, po prelomení psychologickej bariéry, by nás už nemalo prekvapiť, že v prípade potreby môžeme na seba napojiť aj tri, alebo ešte viac exemplárov tej istej tabuľky.

2.6.5 Ešte jeden odvodený údaj

Keď sme vyrábali súčasný viactabuľkový model, z tabuľky `študent` sme rozšafne vyhodili stĺpec `priemer` s dôvetkom, že veď sa dá vypočítať z iných údajov. Teraz je tá chvíľa. Pod študijným priemerom rozumieme ako obvykle aritmetický priemer hodnotení z absolvovaných predmetov. K tabuľke `osoba`, kde sa skrýva meno a priezvisko, teda pripojíme tabuľku `zapísané`, kde máme všetky hodnotenia. (K tomu, že tabuľka `študent` by tu bola zbytočná, sme sa už raz vyjadrili.) Nesmieme pritom zabudnúť zmeniť dátový typ priemeru na desatinné číslo. Takže:

```
SELECT
  o.id,
  o.meno,
  o.priezvisko,
  AVG(DEC(z.hodnotenie)) AS priemer
FROM
  osoba AS o
    JOIN zapísané AS z ON z.id_študent = o.id
GROUP BY
  o.id,
  o.meno,
  o.priezvisko
```

Odpoveď:

ID	MENO	PRIEZVISKO	PRIEMER
101	Ján	Hraško	2,00000000000000000000
102	Ružena	Šípová	3,00000000000000000000
103	Aladár	Baba	2,25000000000000000000
104	Ferdinand	Mravec	2,00000000000000000000
105	Ján	Polienko	1,80000000000000000000
106	Juraj	Truľo	1,50000000000000000000
107	Jana	Botková	1,83333333333333333333
108	Dana	Botková	1,57142857142857142857
109	Ján	Hlúpy	1,60000000000000000000
110	Aladár	Miazga	2,57142857142857142857
111	Mikuláš	Myšiak	1,90909090909090909090
112	Donald	Káčer	2,10000000000000000000
113	Jozef	Námorník	2,60000000000000000000
114	Peter	Pan	1,66666666666666666666

Zdá sa, že zmenou metodiky sa prví stali poslednými a poslední prvými... Buďme preto radi, že sme mechanicky neprevzali pôvodné, ktovie akými bludnými metódami určované študijné priemery.

Všimnime si, že v dopyte nie je iná podmienka okrem väzobnej – pripomeňme, že do priemeru sa zaratúvajú iba neprázdne hodnoty. Ak by sme však chceli do hodnotenia zahrnúť aj predmety, ktoré študent nezvládol (t. j. v zázname je stĺpec `hodnotenie` prázdny, ale `dátum_ukončenia` neprázdny), a to s hodnotou `4`, pomohli by sme si vetvením:

```
SELECT
  o.id,
  o.meno,
  o.priezvisko,
  AVG
  (
    CASE
      WHEN (z.hodnotenie IS NULL AND z.dátum_ukončenia IS NOT NULL) THEN 4
      ELSE DEC(z.hodnotenie)
    END
  ) AS upravený_priemer
FROM
  osoba AS o
    JOIN zapísané AS z ON z.id_študent = o.id
GROUP BY
  o.id,
  o.meno,
  o.priezvisko
```

Odpoveď:

ID	MENO	PRIEZVISKO	UPRAVENÝ_PRIEMER
101	Ján	Hraško	2,00000000000000000000
102	Ružena	Šípová	3,00000000000000000000
103	Aladár	Baba	2,25000000000000000000
104	Ferdinand	Mravec	2,00000000000000000000
105	Ján	Polienko	1,80000000000000000000
106	Juraj	Truľo	1,50000000000000000000
107	Jana	Botková	2,14285714285714285714
108	Dana	Botková	1,57142857142857142857
109	Ján	Hlúpy	2,00000000000000000000
110	Aladár	Miazga	2,57142857142857142857
111	Mikuláš	Myšiak	1,90909090909090909090
112	Donald	Káčer	2,10000000000000000000
113	Jozef	Námorník	2,60000000000000000000
114	Peter	Pan	1,66666666666666666666

2.6.Ú Úlohy

1 Urobte zoznam študentov opakujúcich niektorý predmet.

2 Zistite, či pri zápise nedošlo k nejakému porušeniu (priamych) prerekvizít.

3 Nájdite všetky trojice spolubývajúcich.

4 Zistite, či nebýva v niektorej izbe dvojica osôb rôzneho pohlavia (nemusí to hneď znamenať porušenie dobrých mravov, môžu to byť manželia).

3
Ďalšie črty SQL

3.1 Tranzitívny uzáver a rekurzia

3.1.1 Tranzitívny uzáver

Všimnime si teraz podrobnejšie prerekvizity. Ak máme tri predmety A, B a C také, že A je prerekvizitou B a B je prerekvizitou C, fakticky existuje (z prirodzenej tranzitivity vyplývajúci) prerekvizitný vzťah aj medzi A a C (pričom B má úlohu akéhosi „tranzitu“, prechodu medzi nimi). Napríklad v našich dátach je uvedené, že predmet Databázové systémy musí predchádzať predmetu Fyzika DNA a ten musí predchádzať predmetu Nebeská mechanika. Teda nevyhnutne Databázové systémy sú nutnou podmienkou Nebeskej mechaniky, hoci to v dátach explicitne uvedené nie je. Naša tabuľka `prerekvizita` teda nie je **tranzitívna**. Tabuľku, ktorá z nej vznikne doplnením všetkých nepriamych, vynútených, z tranzitivity vyplývajúcich vzťahov, nazveme jej **tranzitívnym uzáverom**. Ukážeme si, ako ho možno databázovými prostriedkami získať. Použijeme metódu **rekurzie** (čiže, ak chceme, **matematickej indukcie**), spočívajúcej v postupnom iterovaní istého elementárneho kroku.

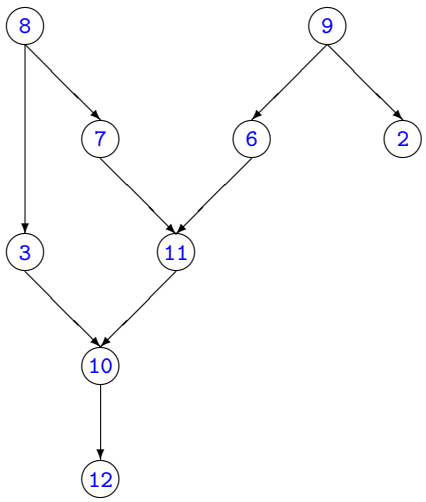
Hľadajme teda tranzitívny uzáver tabuľky `prerekvizita`, ktorá, pripomeňme, vyzerá takto:

```
SELECT *
FROM prerekvizita
```

Odpoveď:

ID_PODMIEŇOVANÝ	ID_PODMIEŇUJÚCI
3	10
6	11
7	11
8	3
8	7
9	2
9	6
10	12
11	10

Priame i nepriame vzťahy si rozdeľme na niekoľko (nie nutne disjunktných) kategórií podľa toho, koľko je medzi nimi prostredníkov. Ak každú dvojicu (identifikátorov) predmetov z pôvodnej tabuľky spojíme hranou orientovanou od hodnoty stĺpca `id_podmieňovaný` k hodnote stĺpca `id_podmieňujúci`, dostaneme takýto (už z logiky veci necyklický) graf:



Do kategórie K_n zaradíme také dvojice, medzi ktorými existuje orientovaná cesta s n hranami. Napríklad dvojica $(3, 12)$ je v kategórii K_2, lebo medzi nimi existuje cesta 3-10-12, a dvojica $(8, 12)$ je v kategórii K_3, lebo medzi nimi existuje cesta 8-3-10-12, ale aj v kategórii K_4 pre cestu 8-7-11-10-12. Pokúsme sa teraz vyjadriť postupne jednotlivé kategórie. K_1 je vlastne priamo pôvodná tabuľka `prerekvizita`. Prvý krok rekurzie teda vyzerá takto:

```
SELECT
  id_podmieňovaný AS z,
  id_podmieňujúci AS do
FROM prerekvizita
```

Odpoveď:

Z	DO
3	10
6	11
7	11
8	3
8	7
9	2
9	6
10	12
11	10

Ak vezmeme dva exempláre pôvodnej tabuľky a spojíme ich tak, že podmieňujúci predmet z prvej bude podmieňovaným predmetom z druhej, dostaneme všetky možné cesty s dvoma hranami. Pripomeňme, že tieto dva exempláre musíme rozlíšiť aliasmi:

```
SELECT
  p1.id_podmieňovaný AS z,
  p1.id_podmieňujúci AS prechod,
  p2.id_podmieňovaný AS prechod,
  p2.id_podmieňujúci AS do
FROM
  prerekvizita AS p1
    JOIN prerekvizita AS p2 ON p1.id_podmieňujúci = p2.id_podmieňovaný
```

Odpoveď:

Z	PRECHOD	PRECHOD	DO
8	3	3	10
9	6	6	11
8	7	7	11
3	10	10	12
11	10	10	12
6	11	11	10
7	11	11	10

Ich začiatky a konce teda budú tvoriť dvojice z kategórie K_2:

```
SELECT
  p1.id_podmieňovaný AS z,
  p2.id_podmieňujúci AS do
FROM
  prerekvizita AS p1
    JOIN prerekvizita AS p2 ON p1.id_podmieňujúci = p2.id_podmieňovaný
```

Odpoveď:

Z	DO
8	10
9	11
8	11
3	12
11	12
6	10
7	10

Získať K_3 už teda nebude problém – tentoraz vezmeme tri exempláre pôvodnej tabuľky a opäť ich zreťazíme:

```
SELECT
  p1.id_podmieňovaný AS z,
  p1.id_podmieňujúci AS prechod12,
  p2.id_podmieňovaný AS prechod12,
  p2.id_podmieňujúci AS prechod23,
  p3.id_podmieňovaný AS prechod23,
  p3.id_podmieňujúci AS do
FROM
  prerekvizita AS p1
    JOIN prerekvizita AS p2 ON p1.id_podmieňujúci = p2.id_podmieňovaný
    JOIN prerekvizita AS p3 ON p2.id_podmieňujúci = p3.id_podmieňovaný
```

Odpoveď:

Z	PRECHOD12	PRECHOD12	PRECHOD23	PRECHOD23	DO
8	3	3	10	10	12
6	11	11	10	10	12
7	11	11	10	10	12
9	6	6	11	11	10
8	7	7	11	11	10

Takže K_3 je:

```
SELECT
  p1.id_podmieňovaný AS z,
  p3.id_podmieňujúci AS do
FROM
  prerekvizita AS p1
    JOIN prerekvizita AS p2 ON p1.id_podmieňujúci = p2.id_podmieňovaný
      JOIN prerekvizita AS p3 ON p2.id_podmieňujúci = p3.id_podmieňovaný
```

Odpoveď:

Z	DO
8	12
6	12
7	12
9	10
8	10

Iste už neprekvapí, že na K_4 potrebujeme štyri exempláre:

```
SELECT
  p1.id_podmieňovaný AS z,
  p1.id_podmieňujúci AS prechod12,
  p2.id_podmieňovaný AS prechod12,
  p2.id_podmieňujúci AS prechod23,
  p3.id_podmieňovaný AS prechod23,
  p3.id_podmieňujúci AS prechod34,
  p4.id_podmieňovaný AS prechod34,
  p4.id_podmieňujúci AS do
FROM
  prerekvizita AS p1
    JOIN prerekvizita AS p2 ON p1.id_podmieňujúci = p2.id_podmieňovaný
      JOIN prerekvizita AS p3 ON p2.id_podmieňujúci = p3.id_podmieňovaný
        JOIN prerekvizita AS p4 ON p3.id_podmieňujúci = p4.id_podmieňovaný
```

Odpoveď:

Z	PRECHOD12	PRECHOD12	PRECHOD23	PRECHOD23	PRECHOD34	PRECHOD34	DO
9	6	6	11	11	10	10	12
8	7	7	11	11	10	10	12

Teda K_4 je:

```
SELECT
  p1.id_podmieňovaný AS z,
  p4.id_podmieňujúci AS do
FROM
  prerekvizita AS p1
    JOIN prerekvizita AS p2 ON p1.id_podmieňujúci = p2.id_podmieňovaný
      JOIN prerekvizita AS p3 ON p2.id_podmieňujúci = p3.id_podmieňovaný
        JOIN prerekvizita AS p4 ON p3.id_podmieňujúci = p4.id_podmieňovaný
```

Odpoveď:

Z	DO
9	12
8	12

Analogicky môžeme vyjadriť K_5, to je však už prázdne:

```
SELECT
  p1.id_podmieňovaný AS z,
  p5.id_podmieňujúci AS do
FROM
  prerekvizita AS p1
    JOIN prerekvizita AS p2 ON p1.id_podmieňujúci = p2.id_podmieňovaný
      JOIN prerekvizita AS p3 ON p2.id_podmieňujúci = p3.id_podmieňovaný
        JOIN prerekvizita AS p4 ON p3.id_podmieňujúci = p4.id_podmieňovaný
          JOIN prerekvizita AS p5 ON p4.id_podmieňujúci = p5.id_podmieňovaný
```

Odpoveď:

Z	DO

Tu môžeme skončiť, lebo ak neexistuje žiadna cesta dĺžky 5, nemôže existovať ani cesta väčšej dĺžky. Hľadaný tranzitívny uzáver tabuľky `prerekvizita` je teda zjednotením kategórií K_1, K_2, K_3 a K_4, môžeme preto napísať výsledný dopyt (hoci správne by sme mali odstrániť duplicity):

```
SELECT
  id_podmieňovaný AS z,
  id_podmieňujúci AS do
FROM prerekvizita

UNION ALL

SELECT
  p1.id_podmieňovaný AS z,
  p2.id_podmieňujúci AS do
FROM
  prerekvizita AS p1
    JOIN prerekvizita AS p2 ON p1.id_podmieňujúci = p2.id_podmieňovaný

UNION ALL
```

```
SELECT
  p1.id_podmieňovaný AS z,
  p3.id_podmieňujúci AS do
FROM
  prerekvizita AS p1
    JOIN prerekvizita AS p2 ON p1.id_podmieňujúci = p2.id_podmieňovaný
      JOIN prerekvizita AS p3 ON p2.id_podmieňujúci = p3.id_podmieňovaný

UNION ALL

SELECT
  p1.id_podmieňovaný AS z,
  p4.id_podmieňujúci AS do
FROM
  prerekvizita AS p1
    JOIN prerekvizita AS p2 ON p1.id_podmieňujúci = p2.id_podmieňovaný
      JOIN prerekvizita AS p3 ON p2.id_podmieňujúci = p3.id_podmieňovaný
        JOIN prerekvizita AS p4 ON p3.id_podmieňujúci = p4.id_podmieňovaný
```

Odpoveď (kvôli prehľadnosti sú tieto štyri medzivýsledky oddelené zdvojenými čiarami):

Z	DO
3	10
6	11
7	11
8	3
8	7
9	2
9	6
10	12
11	10
8	10
9	11
8	11
3	12
11	12
6	10
7	10
8	12
6	12
7	12
9	10
8	10
9	12
8	12

Stačia teda štyri iterácie.

3.1.2 Vyjadrenie tranzitívneho uzáveru pomocou rekurzie

No dobre, ale čo keď sa raz náhodou predmet s číslom 8 stane prerekvizitou predmetu s číslom 1? To by znamenalo, že aj kategória K_5 je neprázdna, a náš dopyt by prestal byť platný. Pravda, vieme ho rýchlo opraviť, ale opäť bude platný len dotiaľ, kým niekto nevyrobí ďalšiu vhodnú prerekvizitu. Lepšie by preto bolo napísať taký dopyt, ktorý by fungoval pre každú (rozumnú) konfiguráciu prerekvizít. Ako na to? Predpokladajme, že máme pre každú kategóriu K_i pripravenú (zatiaľ prázdnu) tabuľku k_i s dvoma stĺpcami z a do typu INT. Najprv vybavme špeciálny prípad, už spomínaný prvý krok indukcie. Vieme už, že k_1 sa (až na názvy stĺpcov) zhoduje s pôvodnou tabuľkou prerekvizita, čo môžeme vyjadriť takto:

```
INSERT INTO k_1
SELECT *
FROM prerekvizita
```

Parafrázujme výrok Lao-c', že „Cesta dlhá tisíc míľ sa začína prvým krokom...“ takto: „Cesta dlhá $n+1$ sa začína prvým krokom... – a zvyšok má dĺžku n“. Ak teda mám dvojicu (a, c) z kategórie K_{n+1}, musí existovať prvok b taký, že dvojica (a, b) je v pôvodnej tabuľke `prerekvizita` a dvojica (b, c) je v kategórii K_n.

Konkrétne kategória K_2 je tvorená prvým a posledným stĺpcom spojenia pôvodnej tabuľky `prerekvizita` a kategórie K_1 (čiže tabuľky `k_1`), ktoré sme už raz (hoci v odlišnej notácii) videli. Úlohu a tu hrá stĺpec `prerekvizita.id_podmieňovaný`, úlohu c stĺpec `k_1.do` a tranzit b je tu `prerekvizita.id_podmieňujúci` a zároveň `k_1.z` (ich rovnosť sme zabezpečili v podmienke):

```
SELECT
  prerekvizita.id_podmieňovaný AS a,
  prerekvizita.id_podmieňujúci AS b,
  k_1.z AS b,
  k_1.do AS c
FROM
  prerekvizita
    JOIN k_1 ON prerekvizita.id_podmieňujúci = k_1.z
```

Odpoveď:

A	B	B	C
8	3	3	10
9	6	6	11
8	7	7	11
3	10	10	12
11	10	10	12
6	11	11	10
7	11	11	10

Takže `k_2` naplníme takto:

```
INSERT INTO k_2
SELECT
  prerekvizita.id_podmieňovaný,
  k_1.do
FROM
  prerekvizita
    JOIN k_1 ON prerekvizita.id_podmieňujúci = k_1.z
```

Skoro rovnako je to s kategóriou K_3, ktorá vznikne ako prvý a posledný stĺpec spojenia pôvodnej tabuľky `prerekvizita` s práve naplnenou tabuľkou `k_2`. Aj tu hrá úlohu a stĺpec `prerekvizita.id_podmieňovaný`, úlohu c tentoraz `k_2.do` a prechod b je tu opäť `prerekvizita.id_podmieňujúci` a zároveň (analogicky k predošlému) `k_2.z`:

```
SELECT
  prerekvizita.id_podmieňovaný AS a,
  prerekvizita.id_podmieňujúci AS b,
  k_2.z AS b,
  k_2.do AS c
FROM
  prerekvizita
    JOIN k_2 ON prerekvizita.id_podmieňujúci = k_2.z
```

Odpoveď:

A	B	B	C
8	10	10	12
6	11	11	12
7	11	11	12
8	7	7	10
9	6	6	10

`k_3` teda naplníme príkazom:

```
INSERT INTO k_3
SELECT
  prerekvizita.id_podmieňovaný,
  k_2.do
FROM
  prerekvizita
    JOIN k_2 ON prerekvizita.id_podmieňujúci = k_2.z
```

A analogicky postupujeme pri ostatných kategóriách. K_{n+1} vznikne z prvého a posledného stĺpca spojenia pôvodnej tabuľky `prerekvizita` s práve naplnenou tabuľkou `k_`n (t. j. kategóriou K_n). Opäť hrá úlohu a stĺpec `prerekvizita.id_podmieňovaný`, úlohu c zas stĺpec `k_`n`.do` a prechod b je tu znova na jednej strane `prerekvizita.id_podmieňujúci` a zároveň na strane druhej `k_`n`.z`. Tabuľku `k_`$(n+1)$ teda naplníme príkazom:

```
INSERT INTO k_(n + 1)
SELECT
  prerekvizita.id_podmieňovaný,
  k_n.do
FROM
  prerekvizita
    JOIN k_n ON prerekvizita.id_podmieňujúci = k_n.z
```

Uvedomme si, že je to vlastne druhý indukčný krok. Problém však je, že takéto tabuľky `k_`n nemáme, a ani mať nemôžeme – nevedeli by sme, koľko ich dopredu predpripraviť. Najlepšie by bolo ukladať si priebežné výsledky (t. j. po prvom kole `k_1`, po druhom `k_1 UNION ALL k_2`, po treťom `k_1 UNION ALL k_2 UNION ALL k_3` atď.) do sumárnej tabuľky (nazvime ju `sumár_k`), v ktorej bude na konci uložený celkový výsledok (avšak včítane prípadných duplicít). DB2 takýto spôsob konštrukcie umožňuje, a to použitím už spomínanej klauzuly `WITH`, ktorá vo vnútornom dopyte pripúšťa meno práve definovanej tabuľky, čím zabezpečíme požadovanú rekurziu. Žiaľ, zápis využívajúci `JOIN` nie je povolený (čo možno považovať za ďalší z nedostatkov DB2), preto podmienky spojenia musíme presunúť do časti `WHERE`. V našom prípade teda napíšeme takýto dopyt:

```
WITH sumár_k (z, do) AS
(
  SELECT *
  FROM prerekvizita

  UNION ALL

  SELECT
    prerekvizita.id_podmieňovaný,
    sumár_k.do
  FROM
    prerekvizita,
    sumár_k
  WHERE prerekvizita.id_podmieňujúci = sumár_k.z
)
SELECT *
FROM sumár_k
```

Jeho jednotlivé iterácie (z ktorých sa však dozvieme až poslednú) potom sú:

Z	DO
3	10
6	11
7	11
8	3
8	7
9	2
9	6
10	12
11	10
8	10
9	11
8	11
3	12
11	12
6	10
7	10

Z	DO
3	10
6	11
7	11
8	3
8	7
9	2
9	6
10	12
11	10
8	10
9	11
8	11
3	12
11	12
6	10
7	10
8	12
6	12
7	12
9	10
8	10

Z	DO
3	10
6	11
7	11
8	3
8	7
9	2
9	6
10	12
11	10
8	10
9	11
8	11
3	12
11	12
6	10
7	10
8	12
6	12
7	12
9	10
8	10
9	12
8	12

Z	DO
3	10
6	11
7	11
8	3
8	7
9	2
9	6
10	12
11	10
8	10
9	11
8	11
3	12
11	12
6	10
7	10
8	12
6	12
7	12
9	10
8	10
9	12
8	12

Výsledok je teda totožný s výsledkom predchádzajúceho postupu. Všimnime si, že vnútorný dopyt pozostáva z dvoch častí – prvá zodpovedá prvému a druhá druhému indukčnému kroku. Takto definovaný vnútorný dopyt sa vzhľadom na rekurziu iteruje, pričom sa vytváraná tabuľka postupne zväčšuje (v každej iterácii sa nejaké záznamy pridajú), a to až dovtedy, kým sa proces pridávania nezastaví, čo je pre systém signál, že rekurziu možno ukončiť (preto je posledná iterácia rovná predposlednej). Počet iterácií dopredu nie je známy (záleží od konkrétneho obsahu pôvodnej tabuľky). Keďže môže byť teoreticky i nekonečný, systém na to pri výpise dát upozorní. V porovnaní s klasickou matematickou indukciou je tu rozdiel v tom, že prvý indukčný krok je zahrnutý do každej iterácie. Podobne v druhom kroku sme jednu hladinu k_n nahradili celou sumárnou tabuľkou sumár_k. Mohlo by sa preto zdať, že sa niektoré kroky zbytočne opakujú, systém však zabezpečuje, aby sa pri každej iterácii tie kroky, ktoré nemôžu priniesť nové záznamy, už nevykonávali. V každej iterácii sa tak v skutočnosti pracuje len so záznamami z hladiny k_n.

Zatiaľ sme pracovali so suchými číslami, skúsme preto praktickejšiu ukážku. Pokúsme sa zistiť, pre ktoré predmety sú (priamou alebo nepriamou) prerekvizitou naše obľúbené Databázové systémy. Všimnime si, že najťažšiu časť – rekurziu pomocou WITH – už máme hotovú, stačí už iba upraviť záverečný dopyt: Identifikátory predmetov nahradíme ich názvami tak, že na oba stĺpce pomocnej tabuľky napojíme osobitný exemplár tabuľky predmet. Kvôli prehľadnosti ešte zmeňme v tejto chvíli už nič nehovoriace názvy sumár_k, z a do na priliehavejšie nepriama_prerekvizita, id_podmieňovaný a id_podmieňujúci. Dostávame tak dopyt:

```
WITH nepriama_prerekvizita (id_podmieňovaný, id_podmieňujúci) AS
(
  SELECT *
  FROM prerekvizita

  UNION ALL

  SELECT p.id_podmieňovaný, n.id_podmieňujúci
  FROM
    prerekvizita AS p,
    nepriama_prerekvizita AS n
  WHERE p.id_podmieňujúci = n.id_podmieňovaný
)
SELECT DISTINCT p1.názov AS predmet_s_prerekvizitou_DBS
FROM
  nepriama_prerekvizita AS n
    JOIN predmet AS p1 ON n.id_podmieňovaný = p1.id
    JOIN predmet AS p2 ON n.id_podmieňujúci = p2.id
WHERE p2.názov = 'Databázové systémy'
```

Odpoveď:

PREDMET_S_PREREKVIZITOU_DBS
Nebeská mechanika
Molekulový modeling
Romológia
Teória hromadnej obsluhy
Teória hier
Fyzika DNA
Fázové prechody a kritické javy

Aj tu vidíme, že bez databázových systémov sa v živote jednoducho nezaobídeme ;-).

3.1.Ú Úlohy

1 Zistite, či pri zápise nedošlo k nejakému porušeniu akejkoľvek prerekvizity (včítane implicitných).

2 Napíšte dopyt, ktorého výsledok bude obsahovať meno a priezvisko každého učiteľa n-krát za sebou, pričom týchto n riadkov bude očíslovaných obvyklým spôsobom.

3 Nech je zvykom, že každý študent v novom roku dedí učebnice po niektorom (služobne staršom) kolegovi (čo by sme mohli modelovať v tabuľke `študent` pridaním stĺpca `id_zdedené_od` typu `INT` ukazujúceho na niektorý iný riadok tabuľky `študent`). Zistite, cez ruky ktorých kolegov prešla kniha, ktorú má teraz požičanú daný študent.

4 Byrokrati vydali nové nariadenie, že študent musí na úrad práce do piatich pracovných dní nahlásiť úspešné absolvovanie skúšky, inak mu skrátia štipendium. Okrem sobôt a nedieľ sa za pracovné dni nerátajú sviatky, ktoré sú povedzme evidované v tabuľke `sviatok`. Napíšte dopyt, ktorý pre daný deň určí príslušnú lehotu.

3.2 Pohľady

3.2.1 Pojem pohľadu

Aj keď rozmiestnenie údajov o študentoch (alebo učiteľoch) do viacerých tabuliek má nesporné výhody, je asi dosť otravné v každom dopyte k tabuľke `student` napájať ostatné tabuľky – podľa potreby `osoba`, `izba`, `krajina`, ... Nemožno nejakým spôsobom vrátiť staré dobré časy, keď bolo všetko pohromade a stačilo pracovať s jednou tabuľkou? Ale iste, možno – riešením je vytvorenie **pohľadu** (často sa používa aj pôvodné anglické slovo **view**). Ten sa, hoci nemá svoje vlastné dáta, tvári ako klasická tabuľka, takže si pri dopytovaní rozdiel medzi nimi ani nevšimneme. Ako ho teda vytvoriť? Príkaz sa, pochopiteľne, začína slovným spojením `CREATE VIEW` („vytvor pohľad“) nasledovaným (vzhľadom na celú databázu jednoznačným) názvom pohľadu (za ktorým môže byť v zátvorkách zoznam názvov stĺpcov) a klasickým slovíčkom `AS`. Napokon sa napíše vnorený dopyt, ktorý pohľad definuje. Najprv však napíšme ten vnútorný dopyt. Trebárs takto:

```
SELECT
  o.id,
  o.meno,
  o.priezvisko,
  o.pohlavie,
  o.dátum_narodenia,
  š.ročník,
  s.kód AS skupina,
  k.názov AS krajina,
  i.číslo AS izba
FROM
  študent AS š
    JOIN osoba AS o ON š.id = o.id
    JOIN študijná_skupina AS s ON s.id = š.id_skupina
    LEFT OUTER JOIN krajina AS k ON k.id = š.id_bydlisko
    LEFT OUTER JOIN izba AS i ON i.id = š.id_izba
ORDER BY 1
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	SKUPINA	KRAJINA	IZBA
101	Ján	Hraško	muž	1987-07-12	1	M	Za siedmimi horami a siedmimi dolami	NULL
102	Ružena	Šípová	žena	1984-02-01	1	M	Za siedmimi horami a siedmimi dolami	101A
103	Aladár	Baba	muž	1980-01-22	2	I	Kalifát Bagdad	354B
104	Ferdinand	Mravec	muž	1984-03-03	3	I	Mravenisko	242B
105	Ján	Polienko	muž	1982-04-14	5	MI	Za siedmimi horami a siedmimi dolami	242B
106	Juraj	Truľo	muž	1979-07-16	1	MI	Za siedmimi horami a siedmimi dolami	242B
107	Jana	Botková	žena	1977-09-21	4	MI	NULL	101A
108	Dana	Botková	žena	1977-09-21	4	MI	NULL	101A
109	Ján	Hlúpy	muž	1972-04-01	2	I	Za siedmimi horami a siedmimi dolami	013A
110	Aladár	Miazga	muž	1987-12-22	3	I	NULL	354B
111	Mikuláš	Myšiak	muž	1983-06-06	5	M	Hollywood	321A
112	Donald	Káčer	muž	1982-10-07	5	M	Hollywood	321A
113	Jozef	Námorník	muž	1981-09-23	2	M	NULL	NULL
114	Peter	Pan	muž	2001-01-13	1	I	Neverland	NULL

Takže žiadaný pohľad bude:

```
CREATE VIEW študent_údaje AS
SELECT
  o.id,
  o.meno,
  o.priezvisko,
  o.dátum_narodenia,
  o.pohlavie,
  s.kód AS skupina,
  k.názov AS krajina,
  i.číslo AS izba
FROM
  študent AS š
    JOIN osoba AS o ON š.id = o.id
    JOIN študijná_skupina AS s ON s.id = š.id_skupina
    LEFT OUTER JOIN krajina AS k ON k.id = š.id_bydlisko
    LEFT OUTER JOIN izba AS i ON i.id = š.id_izba
```

alebo alternatívne so zoznamom názvov stĺpcov:

```
CREATE VIEW študent_údaje (id, meno, priezvisko, dátum_narodenia, pohlavie, skupina, krajina, izba, skupina) AS
SELECT
  o.id,
  o.meno,
  o.priezvisko,
  o.dátum_narodenia,
  o.pohlavie,
  s.kód,
  k.názov,
  i.číslo
FROM
  študent AS š
    JOIN osoba AS o ON š.id = o.id
    JOIN študijná_skupina AS s ON s.id = š.id_skupina
    LEFT OUTER JOIN krajina AS k ON k.id = š.id_bydlisko
    LEFT OUTER JOIN izba AS i ON i.id = š.id_izba
```

A teraz ho použijeme, akoby to bola obyčajná tabuľka, pričom odpoveď bude totožná s predchádzajúcou:

```
SELECT *
FROM študent_údaje
ORDER BY 1
```

Dá sa, samozrejme, použiť aj v zložitejšom dopyte. Napríklad ak chceme zistiť predmety, ktoré si študent zapísal, ale ešte ich neabsolvoval, stačí napísať dopyt:

```
SELECT
  š.id,
  š.meno,
  š.priezvisko,
  p.názov AS predmet
FROM
  študent_údaje AS š
    JOIN zapísané AS z ON z.id_študent = š.id
      JOIN predmet AS p ON p.id = z.id_predmet
WHERE z.hodnotenie IS NULL
```

Odpoveď:

ID	MENO	PRIEZVISKO	PREDMET
101	Ján	Hraško	Zložité systémy
101	Ján	Hraško	Databázové systémy
102	Ružena	Šípová	Databázové systémy
103	Aladár	Baba	Sociálnopsychologický výcvik zvládania záťažových situácií
103	Aladár	Baba	Fyzika DNA
105	Ján	Polienko	Romológia
105	Ján	Polienko	Teória hier
106	Juraj	Truľo	Zložité systémy
107	Jana	Botková	Hlbokonepružný rozptyl leptónov na hadrónoch
107	Jana	Botková	Hlbokonepružný rozptyl leptónov na hadrónoch
107	Jana	Botková	Teória hromadnej obsluhy
107	Jana	Botková	Fázové prechody a kritické javy
108	Dana	Botková	Fázové prechody a kritické javy
109	Ján	Hlúpy	Zložité systémy
109	Ján	Hlúpy	Zložité systémy
110	Aladár	Miazga	Teória hromadnej obsluhy
110	Aladár	Miazga	Databázové systémy
111	Mikuláš	Myšiak	Romológia
112	Donald	Káčer	Romológia
112	Donald	Káčer	Teória hier
113	Jozef	Námorník	Zložité systémy
113	Jozef	Námorník	Hlbokonepružný rozptyl leptónov na hadrónoch
113	Jozef	Námorník	Fyzika DNA

Uvedomme si jednu dôležitú vec – pohľad promptne reaguje na zmenu dát v tabuľkách, pomocou ktorých je vytvorený. Vložme do systému novú študentku:

```
INSERT INTO osoba
VALUES (115, 'Kristína', 'Miazgová', 'žena', '11.11.1982')
;
INSERT INTO študent
VALUES (115, 1, 1, NULL, 1)
```

Zmenili sme obsah tabuliek `osoba` a `študent`, a preto sa automaticky zmení i obsah pohľadu `študent_údaje`:

```
SELECT *
FROM študent_údaje
ORDER BY 1
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	SKUPINA	KRAJINA	IZBA
101	Ján	Hraško	muž	1987-07-12	1	M	Za siedmimi horami a siedmimi dolami	NULL
102	Ružena	Šípová	žena	1984-02-01	1	M	Za siedmimi horami a siedmimi dolami	101A
103	Aladár	Baba	muž	1980-01-22	2	I	Kalifát Bagdad	354B
104	Ferdinand	Mravec	muž	1984-03-03	3	I	Mravenisko	242B
105	Ján	Polienko	muž	1982-04-14	5	MI	Za siedmimi horami a siedmimi dolami	242B
106	Juraj	Truľo	muž	1979-07-16	1	MI	Za siedmimi horami a siedmimi dolami	242B
107	Jana	Botková	žena	1977-09-21	4	MI	NULL	101A
108	Dana	Botková	žena	1977-09-21	4	MI	NULL	101A
109	Ján	Hlúpy	muž	1972-04-01	2	I	Za siedmimi horami a siedmimi dolami	013A
110	Aladár	Miazga	muž	1987-12-22	3	I	NULL	354B
111	Mikuláš	Myšiak	muž	1983-06-06	5	M	Hollywood	321A
112	Donald	Káčer	muž	1982-10-07	5	M	Hollywood	321A
113	Jozef	Námorník	muž	1981-09-23	2	M	NULL	NULL
114	Peter	Pan	muž	2001-01-13	1	I	Neverland	NULL
115	Kristína	Miazgová	žena	1982-11-11	1	M	NULL	101A

Po jej odchode zo školy, t. j. po vykonaní príkazu (pripomeňme, že vzhľadom na kaskádový cudzí kľúč medzi tabuľkami `osoba` a `študent` stačí jeden):

```
DELETE FROM osoba
WHERE id = 115
```

sa obsah tohto pohľadu vráti do pôvodnej podoby.

3.2.2 Hierarchické pohľady

Pohľad sa môže vyskytnúť v definícii iného pohľadu. Ak napríklad chceme len údaje o internátnikoch, napíšeme:

```
CREATE VIEW internátnik_údaje AS
SELECT *
FROM študent_údaje
WHERE izba IS NOT NULL
```

Vypíšeme jeho obsah:

```
SELECT *
FROM internátnik_údaje
ORDER BY 1
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	SKUPINA	KRAJINA	IZBA
102	Ružena	Šípová	žena	1984-02-01	1	M	Za siedmimi horami a siedmimi dolami	101A
103	Aladár	Baba	muž	1980-01-22	2	I	Kalifát Bagdad	354B
104	Ferdinand	Mravec	muž	1984-03-03	3	I	Mravenisko	242B
105	Ján	Polienko	muž	1982-04-14	5	MI	Za siedmimi horami a siedmimi dolami	242B
106	Juraj	Truľo	muž	1979-07-16	1	MI	Za siedmimi horami a siedmimi dolami	242B
107	Jana	Botková	žena	1977-09-21	4	MI	NULL	101A
108	Dana	Botková	žena	1977-09-21	4	MI	NULL	101A
109	Ján	Hlúpy	muž	1972-04-01	2	I	Za siedmimi horami a siedmimi dolami	013A
110	Aladár	Miazga	muž	1987-12-22	3	I	NULL	354B
111	Mikuláš	Myšiak	muž	1983-06-06	5	M	Hollywood	321A
112	Donald	Káčer	muž	1982-10-07	5	M	Hollywood	321A

Takto môžeme dokonca vytvoriť celú hierarchiu pohľadov.

3.2.3 Rekurzívne pohľady

Definícia pohľadu môže byť aj **rekurzívna**, v tom prípade však musí byť vnútorný dopyt v zátvorkách. Vráťme sa k príkladu s implicitnými prerekvizitami a nahraďme pomocnú tabuľku takýmto rekurzívnym pohľadom (všimnime si, že aj tu musíme podmienku spojenia presunúť do časti `WHERE`):

```
CREATE VIEW nepriama_prerekvizita (id_podmieňovaný, id_podmieňujúci) AS
(
  SELECT *
  FROM prerekvizita

  UNION ALL

  SELECT
    p.id_podmieňovaný,
    n.id_podmieňujúci
  FROM
    prerekvizita AS p,
    nepriama_prerekvizita AS n
  WHERE p.id_podmieňujúci = n.id_podmieňovaný
)
```

Vypíšeme jeho obsah:

```
SELECT *
FROM nepriama_prerekvizita
```

Výsledok je nám už známy tranzitívny uzáver:

ID_PODMIEŇOVANÝ	ID_PODMIEŇUJÚCI
3	10
6	11
7	11
8	3
8	7
9	2
9	6
10	12
11	10
8	10
9	11
8	11
3	12
11	12
6	10
7	10
8	12
6	12
7	12
9	10
8	10
9	12
8	12

3.2.4 Ovplyvňovanie tabuľky skrz pohľad

Vzťah medzi pohľadom a tabuľkou môže byť aj opačný – formálnym vymazaním alebo modifikáciou existujúceho či vložením nového záznamu do pohľadu môžeme ovplyvniť aj obsah pôvodnej tabuľky. Musia však byť splnené isté podmienky, základnou je jednoduchosť definície pohľadu – zjednodušene povedané, môže v nej byť použitá (priamo či prostredníctvom iného pohľadu) len jedna tabuľka. Majme napríklad pohľad `žena` definovaný len pomocou tabuľky `osoba` takto:

```
CREATE VIEW žena AS
SELECT *
FROM osoba
WHERE pohlavie = 'žena'
```

Jeho obsah je zrejme:

```
SELECT *
FROM žena
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA
3	Vasilisa	Premúdra	žena	1970-22-01
4	Hedviga	Baba	žena	1784-05-03
102	Ružena	Šípová	žena	1984-02-01
107	Jana	Botková	žena	1977-09-21
108	Dana	Botková	žena	1977-09-21

Novú osobu môžeme potom vložiť aj do tohto pohľadu (samozrejme, len formálne, veď vieme, že pohľad vlastné dáta nemá):

```
INSERT INTO žena
VALUES (115, 'Kristína', 'Miazgová', 'žena', '11.11.1982')
```

Zmení sa tým obsah tabuľky `osoba` (a následne, samozrejme, aj pohľad `žena`):

```
SELECT *
FROM osoba
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA
1	Gejza	Miazga	muž	1955-12-12
2	Matej	Múdry	muž	1945-06-11
3	Vasilisa	Premúdra	žena	1970-22-01
4	Hedviga	Baba	žena	1784-05-03
5	d'Eduard	Vševed	muž	1900-03-13
101	Ján	Hraško	muž	1987-07-12
102	Ružena	Šípová	žena	1984-02-01
103	Aladár	Baba	muž	1980-01-22
104	Ferdinand	Mravec	muž	1984-03-03
105	Ján	Polienko	muž	1982-04-14
106	Juraj	Truľo	muž	1979-07-16
107	Jana	Botková	žena	1977-09-21
108	Dana	Botková	žena	1977-09-21
109	Ján	Hlúpy	muž	1972-04-01
110	Aladár	Miazga	muž	1987-12-22
111	Mikuláš	Myšiak	muž	1983-06-06
112	Donald	Káčer	muž	1982-10-07
113	Jozef	Námorník	muž	1981-09-23
114	Peter	Pan	muž	2001-01-13
115	Kristína	Miazgová	žena	1982-11-11

Tabuľka zareaguje aj na modifikáciu záznamu:

```
UPDATE žena
SET dátum_narodenia = '10.11.1982'
WHERE id = 115
```

Vyzerá potom takto:

```
SELECT *
FROM osoba
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA
1	Gejza	Miazga	muž	1955-12-12
2	Matej	Múdry	muž	1945-06-11
3	Vasilisa	Premúdra	žena	1970-22-01
4	Hedviga	Baba	žena	1784-05-03
5	d'Eduard	Vševed	muž	1900-03-13
101	Ján	Hraško	muž	1987-07-12
102	Ružena	Šípová	žena	1984-02-01
103	Aladár	Baba	muž	1980-01-22
104	Ferdinand	Mravec	muž	1984-03-03
105	Ján	Polienko	muž	1982-04-14
106	Juraj	Truľo	muž	1979-07-16
107	Jana	Botková	žena	1977-09-21
108	Dana	Botková	žena	1977-09-21
109	Ján	Hlúpy	muž	1972-04-01
110	Aladár	Miazga	muž	1987-12-22
111	Mikuláš	Myšiak	muž	1983-06-06
112	Donald	Káčer	muž	1982-10-07
113	Jozef	Námorník	muž	1981-09-23
114	Peter	Pan	muž	2001-01-13
115	Kristína	Miazgová	žena	1982-11-10

A napokon vymazanie:

```
DELETE FROM žena
WHERE id = 115
```

Tabuľka `osoba` potom vyzerá takto:

```
SELECT *
FROM osoba
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA
1	Gejza	Miazga	muž	1955-12-12
2	Matej	Múdry	muž	1945-06-11
3	Vasilisa	Premúdra	žena	1970-22-01
4	Hedviga	Baba	žena	1784-05-03
5	d'Eduard	Vševed	muž	1900-03-13
101	Ján	Hraško	muž	1987-07-12
102	Ružena	Šípová	žena	1984-02-01
103	Aladár	Baba	muž	1980-01-22
104	Ferdinand	Mravec	muž	1984-03-03
105	Ján	Polienko	muž	1982-04-14
106	Juraj	Truľo	muž	1979-07-16
107	Jana	Botková	žena	1977-09-21
108	Dana	Botková	žena	1977-09-21
109	Ján	Hlúpy	muž	1972-04-01
110	Aladár	Miazga	muž	1987-12-22
111	Mikuláš	Myšiak	muž	1983-06-06
112	Donald	Káčer	muž	1982-10-07
113	Jozef	Námorník	muž	1981-09-23
114	Peter	Pan	muž	2001-01-13

Je zaujímavé, že do takto definovaného pohľadu môžeme vložiť aj muža:

```
INSERT INTO žena
VALUES (115, 'Martin', 'Klingáč', 'muž', '1.1.1991')
```

Obsah tabuľky osoba sa tiež zmení:

```
SELECT *
FROM osoba
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA
1	Gejza	Miazga	muž	1955-12-12
2	Matej	Múdry	muž	1945-06-11
3	Vasilisa	Premúdra	žena	1970-22-01
4	Hedviga	Baba	žena	1784-05-03
5	d'Eduard	Vševed	muž	1900-03-13
101	Ján	Hraško	muž	1987-07-12
102	Ružena	Šípová	žena	1984-02-01
103	Aladár	Baba	muž	1980-01-22
104	Ferdinand	Mravec	muž	1984-03-03
105	Ján	Polienko	muž	1982-04-14
106	Juraj	Truľo	muž	1979-07-16
107	Jana	Botková	žena	1977-09-21
108	Dana	Botková	žena	1977-09-21
109	Ján	Hlúpy	muž	1972-04-01
110	Aladár	Miazga	muž	1987-12-22
111	Mikuláš	Myšiak	muž	1983-06-06
112	Donald	Káčer	muž	1982-10-07
113	Jozef	Námorník	muž	1981-09-23
114	Peter	Pan	muž	2001-01-13
115	Martin	Klingáč	muž	1991-01-01

Avšak aj keď sme vkladali (akože) do pohľadu žena, nový záznam v ňom nenájdeme:

```
SELECT *
FROM žena
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA
3	Vasilisa	Premúdra	žena	1970-22-01
4	Hedviga	Baba	žena	1784-05-03
102	Ružena	Šípová	žena	1984-02-01
107	Jana	Botková	žena	1977-09-21
108	Dana	Botková	žena	1977-09-21

Nečudo, veď Martin Klingáč nie je žena. Aby sme takejto transvestii zabránili, musíme definíciu pohľadu jemne zmeniť – za doterajšiu definíciu pridáme slovné spojenie WITH CHECK OPTION („s možnosťou kontroly“):

```
DELETE FROM osoba
WHERE id = 115
;
DROP VIEW žena
;
CREATE VIEW žena AS
SELECT *
FROM osoba
WHERE pohlavie = 'žena'
WITH CHECK OPTION
```

Teraz sa už vložiť žiadneho muža nepodarí – vkladaný záznam totiž nevyhovuje podmienke vo WHERE-klauzule definície pohľadu.

Ovplyvňovanie pôvodnej tabuľky funguje aj cez viac medzistupňov. Ak nad pohľadom žena vytvoríme pohľad dievča príkazom:

```
CREATE VIEW dievča AS
SELECT *
FROM žena
WHERE YEAR(dátum_narodenia) >= 1975
WITH CHECK OPTION
```

čím vznikne hierarchia osoba-žena-dievča, nasledujúci príkaz spôsobí zmenu v tabuľke osoba (a, isteže, aj vo zvyšku tejto postupnosti):

```
INSERT INTO dievča
VALUES (115, 'Kristína', 'Miazgová', 'žena', '11.11.1982')
```

Naozaj, stav tabuľky osoba je:

```
SELECT *
FROM osoba
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA
1	Gejza	Miazga	muž	1955-12-12
2	Matej	Múdry	muž	1945-06-11
3	Vasilisa	Premúdra	žena	1970-22-01
4	Hedviga	Baba	žena	1784-05-03
5	d'Eduard	Vševed	muž	1900-03-13
101	Ján	Hraško	muž	1987-07-12
102	Ružena	Šípová	žena	1984-02-01
103	Aladár	Baba	muž	1980-01-22
104	Ferdinand	Mravec	muž	1984-03-03
105	Ján	Polienko	muž	1982-04-14
106	Juraj	Truľo	muž	1979-07-16
107	Jana	Botková	žena	1977-09-21
108	Dana	Botková	žena	1977-09-21
109	Ján	Hlúpy	muž	1972-04-01
110	Aladár	Miazga	muž	1987-12-22
111	Mikuláš	Myšiak	muž	1983-06-06
112	Donald	Káčer	muž	1982-10-07
113	Jozef	Námorník	muž	1981-09-23
114	Peter	Pan	muž	2001-01-13
115	Kristína	Miazgová	žena	1982-11-11

Nakoniec si všimnime, že tabuľku môžeme v princípe ovplyvňovať aj prostredníctvom (na nej definovaného) pohľadu nezahŕňajúceho všetky jej stĺpce. Vynechané stĺpce však musia byť NOT NULL, resp. mať preddefinovanú hodnotu. Ak teda definujeme pohľad:

```
CREATE VIEW zápis AS
SELECT
  id,
  id_študent,
  id_predmet
FROM zapísané
```

a vložíme „doň" zápis predmetu s číslom 1 Kristínou Miazgovou:

```
INSERT INTO zápis
VALUES (1 + (SELECT VALUE(MAX(id),0) FROM zápis), 115, 1)
```

v tabuľke zapísané sa objaví riadok:

```
SELECT *
FROM zapísané
WHERE id_študent = 115
```

Odpoveď:

ID	ID_ŠTUDENT	ID_PREDMET	DÁTUM_ZÁPISU	DÁTUM_UKONČENIA	HODNOTENIE
106	115	1	dnešný deň	NULL	NULL

3.2.5 Mazanie pohľadu

Prv než zabudneme, podotknime, že pohľad sa zruší už ľahko odvoditeľným príkazom DROP VIEW („strať pohľad“) nasledovaným názvom tabuľky. Po príkaze (pred malou chvíľou sme ho už nevdojak použili):

```
DROP VIEW žena
```

teda pohľad žena už nebudeme môcť použiť. Vôbec to však, samozrejme, neznamená, že sme dáta z pohľadu stratili – veď vieme, že pohľad žiadne „vlastné“ dáta nemá. Rovnako žiadne dáta nezmizli ani z tabuľky osoba, ktorú sme pri definícii tohto pohľadu použili (a tobôž nie zo žiadnej inej).

Všimnime si ešte, ako sa budú pri mazaní správať pohľady z hierarchie – po vymazaní tabuľky alebo pohľadu je celkom prirodzene znemožnená aj práca s pohľadmi, ktorých definícia je na nich založená. Napríklad po vymazaní pohľadu žena nebude použiteľný ani pohľad dievča, ktorý je na ňom vybudovaný. Zo záhadných dôvodov známych len tvorcom DB2 sa však po znovudefinovaní pohľadu žena pohľad dievča nesfunkční, a treba ho tak či tak definovať nanovo.

3.2.6 Načo ešte sú pohľady?

Zjednodušenie dopytov nie je jediným dôvodom zavedenia pohľadov. S databázou často pracuje viacero používateľov a niekedy nie je vhodné, aby všetci videli všetko. Pred niektorými z nich preto možno isté citlivé údaje zatajiť tým, že im nepovolíme priamy prístup k tabuľke (dokonca o nej nemusia ani vedieť), ale iba k jej časti obsahujúcej im prístupné dáta, ktoré sú sprostredkované práve pohľadom. (Napríklad v pohľade zápis sme takto utajili stĺpce dátum_zápisu, dátum_ukončenia a hodnotenie.) Pohľady sa tak často stávajú akýmsi prostredníkom medzi databázou a bežným používateľom.

3.3 Triggery

3.3.1 Ako funguje trigger?

Keď študent úspešne absolvuje predmet, učiteľ mu zapíše (doposiaľ nevyplnené) hodnotenie. Popritom by však bolo vhodné vyplniť doteraz prázdny dátum ukončenia predmetu aktuálnym dňom. Dá sa to, pravdaže, urobiť aj explicitne, keď príkaz

```
UPDATE zapísané
SET hodnotenie = ...
WHERE id = ...
  AND hodnotenie IS NULL
```

nahradíme príkazom

```
UPDATE zapísané
SET
  hodnotenie = ...,
  dátum_ukončenia = CURRENT DATE
WHERE id = ...
  AND hodnotenie IS NULL
```

Hrozí však, že raz to zabudneme urobiť, a naše údaje budú nekonzistentné. Nedalo by sa to nejako zautomatizovať? No, dalo – treba použiť tzv. **trigger** (alebo aj **spúšťač**). Je to mechanizmus, ktorý pri splnení dopredu definovaných podmienok automaticky zabezpečí vykonanie dopredu definovanej **akcie**. V našom prípade je tou podmienkou zmena hodnotenia na neprázdnu hodnotu a spúšťaným príkazom nastavenie dátumu absolvovania predmetu. Vyjadríme to takýmto príkazom:

```
CREATE TRIGGER dátum_hodnotenia
NO CASCADE BEFORE UPDATE OF hodnotenie ON zapísané
REFERENCING
  NEW AS n
  OLD AS o
FOR EACH ROW
MODE DB2SQL
WHEN (o.hodnotenie IS NULL)
SET n.dátum_ukončenia = CURRENT DATE
```

Pár viet k (základnej) syntaxi a významu jednotlivých častí definície triggera: Po kľúčovom slovnom spojení `CREATE TRIGGER` („vytvor spúšťač“) nasleduje meno triggera. Za tým nasledujúce `NO CASCADE BEFORE` („nekaskádovo pred“) hovorí, že akcia sa má spustiť ešte pred zmenou v databáze (nekaskádovosťou rozumieme to, že takto spustená akcia nespôsobí aktiváciu žiadneho iného triggera). Na tomto mieste možno použiť aj slovo `AFTER` („po“), ktoré znamená, že akcia sa vykoná až po zmene v databáze. Nasleduje typ operácie, ktorá vyvoláva akciu. V našom prípade je to `UPDATE`, no prípustné sú aj operácie `INSERT` a `DELETE`. V prípade `UPDATE`-u potom môže po predložke `OF` (väzba na genitív) nasledovať čiarkami oddelený zoznam stĺpcov, čo znamená, že akciu môže vyvolať zmena v ľubovoľnom z nich. V našom prípade ide o jediný stĺpec `hodnotenie`. Na zmenu v nevymenovaných stĺpcoch systém nebude reagovať. Ak tam však predložka `OF` so zoznamom stĺpcov nie je, akcia sa týka zmeny v ľubovoľnom stĺpci. Ďalej po povinnej predložke `ON` („na“, „týkajúci sa“) sa napíše názov tabuľky, ktorá má byť takto citlivá na zmenu. U nás je to tabuľka `zapísané`. Príkaz ďalej môže (ale nemusí) pokračovať sekciou `REFERENCING` („odvolávajúc sa“), označením toho, na čo sa budeme neskôr odvolávať. V našom prípade ide o riadok, ktorý upravujeme, o čom hovorí časť `NEW AS` („nový ako“) nasledovaná naším označením `n` meneného riadka po zmene, ale okrem toho je tam aj `OLD AS` („starý ako“) tiež nasledovaná zvoleným označením `o`, ktorá hovorí o hodnotách záznamu ešte pred zmenou. Ak by sa akcia netýkala jedného riadku, ale celej skupiny záznamov, dalo by sa namiesto toho použiť aj `NEW_TABLE AS` („nová tabuľka ako“) (pre skupinu novovytvorených záznamov alebo záznamov po modifikácii) a/alebo `OLD_TABLE AS` („stará tabuľka ako“) (pre skupinu mazaných záznamov alebo záznamov pred modifikáciou), obe, samozrejme, nasledované zvoleným a neskôr použitým identifikátorom. Ďalšia povinná časť definície triggera hovorí, či sa má akcia vykonať pre každý postihnutý riadok – vtedy sa, ako

v našom prípade, napíše FOR EACH ROW („pre každý (zasiahnutý) riadok“) – alebo iba raz, pre príkaz ako celok – v tom prípade to bude FOR EACH STATEMENT („pre každé vykonanie príkazu“). Časť MODE DB2SQL hovorí o bližšie nešpecifikovanom móde triggera. Je povinná a nemá alternatívu (hmm, prečo sa potom musí písať?). Teraz prichádza nepovinná časť WHEN („kedy“) vyjadrujúca v zátvorkách logickú podmienku, kedy sa má nasledujúca akcia vykonať. Možno v nej okrem stĺpcov tabuľky používať aj hodnoty odkazované pomocou identifikátorov označených v časti REFERENCING – tak ako v našom prípade: o.hodnotenie IS NULL (zmena dátumu nastáva len vtedy, keď bolo pôvodné hodnotenie nevyplnené). Ak časť WHEN chýba, akcia sa spustí vždy. Napokon nasleduje samotná akcia. Môže to byť klasický príkaz modifikácie (UPDATE, INSERT, či DELETE) alebo, ako v našom prípade, priame nastavenie hodnoty stĺpca za slovom SET („nastav“). Vo všetkých týchto prípadoch môžeme použiť ako hodnoty stĺpcov tabuľky, tak odkazovaných záznamov (v našom prípade príkazom n.dátum_ukončenia = CURRENT DATE nastavujeme stĺpec dátum_ukončenia upravovaného riadku n na aktuálny dátum). Za zmienku stojí špeciálny typ akcie SIGNAL SQLSTATE („signalizuj číslo stavu“) nasledovanej číslom stavu v apostrofoch a vysvetľujúcim textom v zátvorkách, ktoré sa vypíšu ako signál, že popísaná situácia nastala. Treba však dodať, že nemožno navzájom kombinovať všetky možnosti, či už z dôvodov ich nekonzistencie, alebo preto, že nie sú v systéme implementované. Ako pars pro toto uveďme, že pri DELETE-e nemá zmysel použiť NEW AS a pri INSERT-e naopak OLD AS. Pokus vytvoriť taký trigger systém jednoducho odmietne.

Ako teda bude fungovať náš vyššie definovaný trigger? Všimnime si najprv, ako vyzerá istá časť tabuľky zapísané:

```
SELECT *
FROM zapísané
WHERE id_študent = 101
```

Odpoveď:

ID	ID_ŠTUDENT	ID_PREDMET	DÁTUM_ZÁPISU	DÁTUM_UKONČENIA	HODNOTENIE
1	101	1	2003-09-01	2004-05-20	2
2	101	4	2003-09-01	NULL	NULL
3	101	12	2003-09-01	NULL	NULL

Po ohodnotení študenta s číslom 101 za predmet s číslom 4, čiže záznamu s identifikátorom 2, známkou 3, a to vykonaním príkazu:

```
UPDATE zapísané
SET hodnotenie = 3
WHERE id = 2
```

sa nezmenil len stĺpec hodnotenie, ale i dátum_ukončenia:

```
SELECT *
FROM zapísané
WHERE id_študent = 101
```

Odpoveď:

ID	ID_ŠTUDENT	ID_PREDMET	DÁTUM_ZÁPISU	DÁTUM_UKONČENIA	HODNOTENIE
1	101	1	2003-09-01	2004-05-20	2
2	101	4	2003-09-01	dnešný dátum	3
3	101	12	2003-09-01	NULL	NULL

Zmena v inom stĺpci dátum_ukončenia neovplyvní. Napríklad po príkaze:

```
UPDATE zapísané
SET id = 999
WHERE id = 3
```

vyzerá príslušná časť tabuľky takto:

```
SELECT *
FROM zapísané
WHERE id_študent = 101
```

Odpoveď:

ID	ID_ŠTUDENT	ID_PREDMET	DÁTUM_ZÁPISU	DÁTUM_UKONČENIA	HODNOTENIE
1	101	1	2003-09-01	2004-05-20	2
2	101	4	2003-09-01	dnešný dátum	3
999	101	12	2003-09-01	NULL	NULL

Stĺpec `dátum_ukončenia` sa nezmení ani vtedy, keď síce upravujeme stĺpec `hodnotenie`, ale ten už bol vyplnený nejakou neprázdnou hodnotou; nie je totiž splnená podmienka vo `WHEN`. Napríklad po prípadnej oprave nesprávne zapísaného hodnotenia príkazom:

```
UPDATE zapísané
SET hodnotenie = 1
WHERE id = 1
```

vyzerá príslušná časť tabuľky takto:

```
SELECT *
FROM zapísané
WHERE id_študent = 101
```

Odpoveď:

ID	ID_ŠTUDENT	ID_PREDMET	DÁTUM_ZÁPISU	DÁTUM_UKONČENIA	HODNOTENIE
1	101	1	2003-09-01	2004-05-20	1
2	101	4	2003-09-01	dnešný dátum	3
999	101	12	2003-09-01	NULL	NULL

Vidíme, že stĺpec `dátum_ukončenia` sa v príslušnom riadku naozaj nezmenil. Vráťme teraz tabuľku do pôvodnej podoby:

```
UPDATE zapísané
SET id = 3
WHERE id = 999
;
UPDATE zapísané
SET hodnotenie = 2
WHERE id = 1
;
UPDATE zapísané
SET
  hodnotenie = NULL,
  dátum_ukončenia = NULL
WHERE id = 2
```

Tu si uvedomme, že ani druhý, ani tretí príkaz náš trigger nevyvolajú, lebo pred jeho vykonaním nie sú príslušné hodnoty stĺpca `hodnotenie` prázdne.

A teraz upravme hodnotenia vo viacerých riadkoch naraz:

```
UPDATE zapísané
SET hodnotenie = 1
WHERE id IN (1, 2, 3)
```

Tabuľka sa zmení takto:

```
SELECT *
FROM zapísané
WHERE id_študent = 101
```

Odpoveď:

ID	ID_ŠTUDENT	ID_PREDMET	DÁTUM_ZÁPISU	DÁTUM_UKONČENIA	HODNOTENIE
1	101	1	2003-09-01	2004-05-20	1
2	101	4	2003-09-01	dnešný dátum	1
3	101	12	2003-09-01	dnešný dátum	1

Vidíme teda, že trigger spustil akciu pre každý modifikovaný riadok spĺňajúci podmienku vo `WHEN`.

Ak by nám toto správanie prestalo vyhovovať, nič nám nebráni trigger z databázy vymazať, a to bez strachu, že by to nejako ovplyvnilo dáta. Asi už nikoho neprekvapí, ako to urobíme – po kľúčovom slovnom spojení `DROP TRIGGER` („strať spúšťač“) nasleduje názov triggera. Napríklad v našom prípade by to bolo:

```
DROP TRIGGER dátum_hodnotenia
```

Ešte poriadok:

```
UPDATE zapísané
SET hodnotenie = 2
WHERE id = 1
;
UPDATE zapísané
SET
  hodnotenie = NULL,
  dátum_ukončenia = NULL
WHERE id IN (2, 3)
```

3.3.2 Triggery signalizujúce chybu

Pri predchádzajúcich úpravách si všimnime, že aj keď sa dátum ukončenia mení automaticky, vieme ho potom dodatočne upraviť. Aby táto obchádzka nemohla byť zneužitá, skúsme zakázať priamu úpravu stĺpca `dátum_ukončenia` (teda jediná možnosť jeho úpravy bude potom vyplnenie stĺpca `hodnotenie`). Ako inak, triggerom. Použijeme pri tom akciu `SIGNAL SQLSTATE`. Takže:

```
CREATE TRIGGER zákaz_zmeny_dh
NO CASCADE BEFORE UPDATE OF dátum_ukončenia ON zapísané
REFERENCING OLD AS o
FOR EACH ROW
MODE DB2SQL
SIGNAL SQLSTATE 'NEROB'
  ('Neopováž sa takto podvodne meniť dáta! Ostane tam '
    || VALUE(CHAR(o.dátum_ukončenia,EUR),'NULL') || '!')
```

Pri neúspešnom pokuse zmeniť `dátum_ukončenia`:

```
UPDATE zapísané
SET dátum_ukončenia = '1.1.1111'
WHERE id = 2
```

systém okrem iného zahlási:

```
... Application raised error with diagnostic text:
"Neopováž sa takto podvodne meniť dáta! Ostane tam NULL!".
SQLSTATE=NEROB
```

a dáta ostanú nezmenené.

Podobne pri pokuse:

```
UPDATE zapísané
SET dátum_ukončenia = '1.1.1111'
WHERE id = 1
```

ostane len pri hláške:

```
... Application raised error with diagnostic text:
"Neopováž sa takto podvodne meniť dáta! Ostane tam 20.05.2004!".
SQLSTATE=NEROB
```

Vysvetľujúci text môže mať najviac 70 znakov a môže obsahovať, ako vidíme, aj výrazy obsahujúce hodnoty stĺpcov.

Namiesto nášho kódu NEROB tam môže byť ľubovoľný päťciferný reťazec cifier (0 až 9) alebo veľkých písmen anglickej abecedy (A až Z). Niektoré z takýchto reťazcov sú už však použité na diagnózu systémom definovaných stavov, a nemožno im preto nanútiť iný vysvetľujúci text. Ide o reťazce začínajúce sa 00, 01, 02, ale i niektoré so začiatkom 1 až 6 či A až H. Stačí teda začať niektorou z cifier 7 až 9 alebo písmenom I až Z, a vzniknutý kód bude pri SQL STATE použiteľný (ak nás, pravdaže, už niekto nepredbehol).

Poďme na ďalší príklad, v ktorom si ukážeme trigger pracujúci s viacerými záznamami naraz, teda používajúci odkaz na OLD_TABLE, resp. NEW_TABLE: Chcime zabezpečiť, aby nebolo možné vyhodiť naraz viac ako päť študentov. Pravdaže, kto chce psa biť, palicu si nájde – túto podmienku môžeme obísť tým, že študentov podelíme do viacerých skupín po piatich členoch a povyhadzujeme ich postupne... Tu však ide skôr o to, aby neboli nejakým omylom z databázy vyhodení všetci študenti, veď stačí tak málo – zabudnúť WHERE... Budeme sa teda rozhodovať podľa počtu riadkov celej tabuľky o vyhadzovaných záznamov, žiaden jednotlivý záznam by nám nepomohol.

Všimnime si ešte, že tento trigger stačí definovať ako príkazový (t. j. s FOR EACH STATEMENT), ale zabral by aj riadkový (t. j. s FOR EACH ROW). Naproti tomu slovo AFTER nemožno nahradiť slovným spojením NO CASCADE BEFORE (a to ani v žiadnom inom triggeri používajúcom odkaz na OLD_TABLE či NEW_TABLE), lebo podmienka pracuje s počtom záznamov v tabuľke o, a tie získame až po vykonaní príkazu DELETE. Za pozornosť tiež stojí, že dopyt vnorený vo vysvetľujúcom texte musí byť v osobitných zátvorkách (takže zdanlivo duplicitné otvárajúce zátvorky po slove CHAR sú opodstatnené):

```
CREATE TRIGGER čistka
AFTER DELETE ON osoba
REFERENCING OLD_TABLE AS o
FOR EACH STATEMENT
MODE DB2SQL
WHEN
(
  (
    SELECT COUNT(*)
    FROM o
  )
  > 5
)
SIGNAL SQLSTATE 'POKOJ'
  ('Upokoj sa, nemôžeš vyhodiť ' || RTRIM(CHAR((SELECT COUNT(*) FROM o))) ||' naraz! Najviac 5!')
```

Pri pokuse zrušiť celé študentské osadenstvo naraz:

```
DELETE
FROM osoba
WHERE id IN (SELECT id FROM študent)
```

systém zahlási:

```
... Application raised error with diagnostic text:
"Upokoj sa, nemôžeš vyhodiť 14 naraz! Najviac 5!".
SQLSTATE=POKOJ
```

Pri menšej čistke však učiteľovi do svedomia nevstupuje:

```
DELETE
FROM osoba
WHERE id IN (101, 102, 103)
```

Po týchto troch študentoch neostalo ani stopy:

```
SELECT *
FROM osoba
WHERE id IN (101, 102, 103)
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA

Asi už netreba zdôrazňovať, že vzhľadom na kaskádový charakter príslušných cudzích kľúčov sa z databázy o týchto troch študentoch vymazali všetky možné i nemožné informácie.

Tento trigger (ako napokon každý, ktorý vo svojej definícii obsahuje `AFTER` a `SIGNAL` zároveň) v sebe obsahuje istý paradox, ktorým môžeme opäť trochu viac nazrieť do práce databázového stroja: Slovo `AFTER` hovorí, že prípadná chyba sa zistí až po vykonaní akcie (v tomto prípade vymazania), napriek tomu sa v takom prípade stav databázy vôbec nezmení. Tak vymažú sa, či nie?

Privádza nás to na myšlienku **transakcií**: Každý príkaz sa vykoná najprv len „na skúšku“, a keď po jeho predbežnom vykonaní nenastane žiaden konflikt s niektorým integritným obmedzením (alebo pri triggeri typu `SIGNAL` splnenie podmienky vo `WHEN`), príkaz sa ex post potvrdí (nastáva tzv. „commit“). Inak sa databáza vráti do stavu pred príkazom a tvári sa akoby nič (hovoríme o „rollback“-u).

3.3.3 Triggery so superakciou

Môže sa stať, že pri splnení tej istej aktivačnej podmienky treba vykonať viacero akcií. Nič nám vtedy nebráni definovať viac nezávislých triggerov iniciovaných touto podmienkou, musíme však pri tom rátať s tým, že tieto triggery sa vykonajú v takom poradí, v akom boli definované, čo nám nemusí vždy vyhovovať. Existuje preto aj iná možnosť, ako túto (niekedy podstatnú) následnosť triggerov zabezpečiť, a to vložiť všetky akcie do jedného triggera. V takom prípade po podmienke vo `WHEN` nasleduje **superakcia**, ohraničená kľúčovými slovami `BEGIN ATOMIC` („začni atomické (akcie)“) a `END` („koniec“) a zložená z jednotlivých **atomických akcií** (v správnom poradí) oddelených bodkočiarkami.

Tu treba povedať, že voľba bodkočiarky ako oddeľovača atomických akcií nevrhá na koordináciu tvorcov DB2 práve najlepšie svetlo. Ak totiž chceme spustiť takýto trigger so superakciou, ešte predtým musíme zabezpečiť, aby oddeľovačom príkazov bol iný znak než bodkočiarka. Inak sa totiž pri bodkočiarke oddeľujúcej prvú a druhú akciu superakcie bude systém mylne domnievať, že príkaz definovania triggera sa už skončil, a pre jeho neúplnosť vyhlási chybu.

3.3.Ú Úlohy

1. Keďže je predmet Databázové systémy prerekvizitou pre mnoho ďalších predmetov, definujte trigger, ktorý zabezpečí jeho automatický zápis hneď pri vstupe študenta na univerzitu.
2. Teraz buďte krutí: Definujte trigger, ktorý po dvojnásobnom neúspešnom ukončení toho istého predmetu vyhodí študenta zo školy.
3. Radšej o poriadnejších študentoch: Triggerom zabezpečte, aby po získaní počtu kreditov, ktorý sa rovná 12-násobku čísla ich ročníka, automaticky postúpili do vyššieho ročníka.

4 Zakážte porušenie prerekvizít v tvrdšej verzii – pri pokuse zapísať predmet (t. j. vložiť záznam do tabuľky `zapísané`) s neabsolvovanými prerekvizitami by malo vyskočiť patričné upozornenie a zápis by nemal byť povolený.

5 Definujte trigger, ktorý znemožní prekročenie kapacity izby.

6 Definujte situáciu, kde časť `FOR EACH STATEMENT` príkazového triggera nemožno ekvivalentne nahradiť slovným spojením časť `FOR EACH ROW` (čím by vznikol trigger záznamový).

3.4 Indexy

Keď chceme v nejakej hrubej knihe vyhľadať miesto, kde sa hovorí o nejakej veci, často neostáva nič iné, len ju prelistovať a dúfať, že nám naň padne zrak a že to podľa zákona schválnosti nebude na poslednej možnej strane. Omnoho rozumnejší spôsob, využívaný prevažne v odbornej literatúre, je použiť register čiže index – spravidla abecedne usporiadaný zoznam hesiel. Číslo pri každom z nich je vlastne odkazom na hľadanú stranu, kde sa už ľahko zorientujeme. Namiesto prechádzania celej knihy teda stačí prezrieť index a potom odkazovanú stranu – úspora času priamo závisí od hrúbky knihy.

Rovnakú myšlienku používajú aj databázové systémy. Aby sme urýchlili vyhľadávanie dát v tabuľke, možno k nej vytvoriť jeden alebo viacej **indexov**. Každý index je založený na jednom alebo viacerých stĺpcoch – tých, ktoré sa najčastejšie vyskytujú vo väzobných či obmedzovacích podmienkach. Vytvoríme ho príkazom, ktorý sa začína slovným spojením `CREATE INDEX` („vytvor index“) nasledovaným názvom indexu (aj ten musí byť jednoznačný vzhľadom na celú databázu). Potom ide slovíčko `ON` („na“), názov tabuľky, na ktorej bude index definovaný, a v zátvorkách nasleduje (často jednočlenný) zoznam stĺpcov, podľa ktorých je index urobený, pričom na ich poradí záleží. Môžeme si to vyskúšať na tabuľke `osoba`:

```
CREATE INDEX i_osoba_pm ON osoba (priezvisko, meno)
```

Index je vybudovaný najprv podľa stĺpca `priezvisko` a potom v rámci jedného priezviska podľa stĺpca `meno`. Vidíme teda, že na poradí stĺpcov v zátvorkách záleží.

Treba však povedať, že vzhľadom na maličký počet dát v našej tabuľke je akýkoľvek index kontraproduktívny. Či má zmysel robiť register pre trojstránkovú brožúrku? Bola by to medvedia služba. Tak si nerobme hanbu, a radšej ten index hneď zrušme. Asi už nikoho neprekvapí akým príkazom – za slovné spojenie `DROP INDEX` („strať index“) napíšeme názov indexu:

```
DROP INDEX i_osoba_pm
```

V tejto súvislosti je vhodné poznamenať, že pri vymazaní tabuľky sa, prirodzene, automaticky vymažú aj všetky na nej definované indexy. Opačne to, našťastie, neplatí, dáta sa po zrušení indexu nemenia.

Index podľa stĺpcov v zátvorkách je spravidla vytvorený v klasickom vzostupnom usporiadaní, no môžeme ho zmeniť aj na zostupné tým, že za názov príslušného stĺpca dopíšeme známe `DESC`. Index podľa veku od najmladšieho by teda vyzeral takto:

```
CREATE INDEX i_osoba_n ON osoba (dátum_narodenia DESC)
```

Ak máme záruku, že kombinácia hodnôt indexovaných stĺpcov je jednoznačná, vyplatí sa použiť jednoznačný index – medzi `CREATE` a `INDEX` treba vložiť známe `UNIQUE`. Napríklad za (nesplnenej) podmienky, že žiadni dvaja spolubývajúci nie sú z tej istej krajiny, by sme mohli definovať takýto index:

```
CREATE UNIQUE INDEX iu_študent_ki ON študent (id_bydlisko, id_izba)
```

Všimnime si, že na rozdiel od primárneho a sekundárnych kľúčov nie je pri jednoznačnom indexe zúčastneným stĺpcom zakázaná prázdna hodnota. Správa sa však ako každá iná hodnota, a preto sa nesmie opakovať ani kombinácia, ktorá ju obsahuje.

Ľahko sa presvedčíme, že na jednej tabuľke môže byť definovaných aj viac indexov. Len jeden z nich však môže byť **zhlukový** – podľa neho sú totiž fyzicky zoradené dáta. Tento index nie je „na konci knihy“, zodpovedá skôr samotným číslam strán. Na konci svojej definície má takýto index slovo `CLUSTER` („zhluk“). Napríklad:

```
CREATE INDEX ic_osoba_pm ON osoba (priezvisko, meno) CLUSTER
```

Systém kontroluje, aby neexistovali dva indexy na tú istú postupnosť stĺpcov (včítane ich poradia a usporiadania). Navyše nepripustí dva zhlukové stĺpce. Nepodarí sa nám dokonca vytvoriť ani index podľa primárneho kľúča – ten totiž vznikne automaticky hneď pri jeho vytvorení.

Je dôležité dodať, že index je definovaný na tabuľke ako takej, nie je teda viazaný na jej aktuálnu podobu – funguje aj vtedy, keď sú dáta v príslušnej tabuľke modifikované. Vzhľadom na túto skutočnosť však narastá čas potrebný na modifikáciu jednotlivých záznamov, pri každej zmene sa automaticky prebudováva aj index. Preto v prípade, že je upravovaných dát veľa, je rozumnejšie pred procesom ich modifikácie príslušný index vymazať, a keď je po všetkom, opäť ho vytvoriť.

Na záver zdôraznime jednu zásadnú vec: Indexy sa týkajú len optimalizácie hľadania odpovede na používateľov dopyt, sú teda len vecou databázového stroja. Vytvára ich spravidla administrátor systému a bežný používateľ o nich ani nemusí vedieť. Podobu jeho dopytu to teda vôbec neovplyvní, akurát odpoveď naň príde, verme, skôr.

3.5 Systémové tabuľky

Predstavme si, že je našou úlohou vytvoriť **metamodel** – databázový model, ktorý by postihoval potreby tvorcov databázových systémov. Nech už databázový model vyzerá akokoľvek, vždy ho možno popisovať tými istými slovami – napríklad „tabuľka“, „stĺpec“, „trigger“, či „index“. Napríklad všetky tabuľky tak tvoria (podľa našich doterajších poznatkov) entitný typ s rovnomenným názvom, o každej z nich totiž môžeme povedať zhruba to isté – aký má názov, kto ju vytvoril, alebo aké má stĺpce a koľko ich je. Jednotlivé tabuľky sú teda inštanciami tohto entitného typu. Medzi takto vzniknutými entitnými typmi môžeme vypozorovať aj isté vzťahy a ich kardinality, napríklad jedna tabuľka má niekoľko stĺpcov, ale každý stĺpec patrí práve jednej tabuľke – medzi entitnými typmi *Tabuľka* a *Stĺpec* je teda vzťah typu 1:n. Keď s týmito entitnými typmi a ich vzťahmi zopakujeme proces popísaný v podkapitole 2.4, dostaneme databázový model popisovanej situácie.

Asi takto uvažovali tvorcovia databázového systému DB2. Výsledok ich úvah reálne existuje a je súčasťou každej jednej databázy. Keď nová databáza vzniká, okamžite po vytvorení (teda ešte pred prvým príkazom používateľa na vytvorenie nejakej tabuľky) obsahuje práve tieto **systémové tabuľky**. Tie sú jej pevnou súčasťou a nemožno ich vymazať. Ich názvy sú (včítane bodky) napríklad `SYSIBM.SYSTABLES` pre tabuľky, `SYSIBM.SYSCOLUMNS` pre stĺpce, `SYSIBM.SYSTRIGGERS` pre triggery, či `SYSIBM.SYSINDEXES` pre indexy (veľké písmená naznačujú, že ich mená nie sú volené používateľom). Údaje v systémových tabuľkách budeme nazývať **metadáta**.

Ak chceme napríklad vypísať názvy všetkých tabuliek vytvorených používateľom `xy`, napíšeme dopyt:

```
SELECT NAME
FROM SYSIBM.SYSTABLES
WHERE
      CREATOR = UPPER('xy')
  AND TYPE = 'T'
```

Za predpokladu, že sme tabuľky z nášho univerzitného systému vytvárali práve ako používateľ `xy`, odpoveď vyzerá takto:

NAME
ČLEN_RADY
DRUH_IZBY
FUNKCIA_V_RADE
IZBA
KRAJINA
OSOBA
PREDMET
PREREKVIZITA
ŠTUDENT
ŠTUDIJNÁ_SKUPINA
TITUL
UČITEĽ
ZAPÍSANÉ

Môžeme tak dostať aj prehľad mien systémových tabuliek, stačí zmeniť hodnotu stĺpca `CREATOR` na `SYSIBM`. Z odpovede môžeme dedukovať, aké asi metadáta sú v týchto tabuľkách uložené (a čo všetko ešte o SQL nevieme):

```
SELECT NAME
FROM SYSIBM.SYSTABLES
WHERE
      CREATOR = 'SYSIBM'
  AND TYPE = 'T'
```

Odpoveď:

NAME
SYSATTRIBUTES
SYSBUFFERPOOLNODES
SYSBUFFERPOOLS
SYSCHECKS
SYSCODEPROPERTIES
SYSCOLAUTH
SYSCOLCHECKS
SYSCOLDIST
SYSCOLGROUPDIST
SYSCOLGROUPS
SYSCOLGROUPSCOLS
SYSCOLOPTIONS
SYSCOLPROPERTIES
SYSCOLUMNS
SYSCOLUSE
SYSCOMMENTS
SYSCONSTDEP
SYSDATATYPES
SYSDBAUTH
SYSDEPENDENCIES
SYSEVENTMONITORS
SYSEVENTS
SYSEVENTTABLES
SYSFUNCMAPOPTIONS
SYSFUNCMAPPARMOPTIONS
SYSFUNCMAPPINGS
SYSHIERARCHIES
SYSINDEXAUTH
SYSINDEXCOLUSE
SYSINDEXES
SYSINDEXEXPLOITRULES
SYSINDEXEXTENSIONMETHODS
SYSINDEXEXTENSIONPARMS
SYSINDEXEXTENSIONS
SYSINDEXOPTIONS
SYSJARCONTENTS
SYSJAROBJECTS
SYSKEYCOLUSE
SYSLIBRARIES
SYSLIBRARYAUTH
SYSLIBRARYBINDFILES
SYSLIBRARYVERSIONS
SYSNAMEMAPPINGS
SYSNODEGROUPDEF
SYSNODEGROUPS
SYSPARTITIONMAPS
SYSPASSTHRUAUTH
SYSPLAN
SYSPLANAUTH
SYSPLANDEP
SYSPREDICATESPECS
SYSPROCOPTIONS
SYSPROCPARMOPTIONS
SYSRELS
SYSROUTINEAUTH
SYSROUTINEPARMS
SYSROUTINEPROPERTIESJAVA
SYSROUTINES
SYSSCHEMAAUTH
SYSSCHEMATA
SYSSECTION
SYSSEQUENCEAUTH
SYSSEQUENCES
SYSSERVEROPTIONS
SYSSERVERS
SYSSTMT
SYSTABAUTH
SYSTABCONST
SYSTABLES
SYSTABLESPACES
SYSTABOPTIONS
SYSTBSPACEAUTH
SYSTRANSFORMS
SYSTRIGGERS
SYSTYPEMAPPINGS
SYSUSERAUTH
SYSUSEROPTIONS
SYSVERSIONS
SYSVIEWDEP
SYSVIEWS
SYSWRAPOPTIONS
SYSWRAPPERS
SYSXMLOBJECTAUTH
SYSXMLOBJECTAUTHPERF
SYSXMLOBJECTPROPERTIES
SYSXMLOBJECTRELDEP
SYSXMLOBJECTS
SYSXMLOBJECTXMLDEP
SYSXMLPHYSICALCOLLECTIONS
SYSXMLQUERIES
SYSXMLRELATIONSHIPS
SYSXMLRSPROPERTIES
SYSXMLSTATS

Táto odpoveď je rovnaká vo všetkých databázach vytvorených systémom DB2 (pravdaže, v rovnakej verzii). V týchto tabuľkách sa nachádza všetko, čo je možné vedieť o modeli databázy. Je pekným príkladom **samovzťažnosti**, že jedným z riadkov odpovede je tabuľka SYSTABLES, z ktorej robíme dopyt; obsahuje totiž (okrem iného) aj údaje o sebe samej. Časť názvu tabuľky SYSIBM pred bodkou sa nazýva **schéma** (zoskupenie príbuzných tabuliek), a ak nie je určená inak, obvykle býva totožná s menom používateľa, ktorý ju vytvoril.

Skúsme ešte vypísať stĺpce tabuľky osoba spolu s ich dátovými typmi a schopnosťou prijímať prázdnu hodnotu. Ak nás odpoveď prekvapí, iste len tým, že názvy stĺpcov sú uložené veľkými písmenami (keďže sme ich nedefinovali v úvodzovkách), prípadne tým, že dátové typy INTEGER a DATE majú tiež určenú dĺžku:

```
SELECT
  NAME,
  COLTYPE,
  LENGTH,
  NULLS
FROM SYSIBM.SYSCOLUMNS
WHERE TBNAME = UPPER('osoba')
```

Odpoveď:

NAME	COLTYPE	LENGTH	NULLS
ID	INTEGER	4	N
MENO	VARCHAR	10	N
PRIEZVISKO	VARCHAR	15	N
POHLAVIE	CHAR	4	N
DÁTUM_NARODENIA	DATE	4	N

3.6 Základy administrácie databázy

3.6.1 Vytvorenie a zrušenie databázy

Na istom stupni intelektuálneho vývoja vždy vyvstáva zásadná otázka, kto a akým spôsobom vyslovil ono „Fiat lux!“. Ani v našom databázovom svete to nie je inak. Veselo si v ňom vytvárame tabuľky, vkladáme do nich dáta, modifikujeme a mažeme ich, dopytujeme sa na ne, dokonca sme pokročili tak ďaleko, že ich tranzitívne uzavierame. Asi sme teda dosť zrelí, aby sme pochopili, že priestor, v ktorom sa pohybujeme, tu neexistoval odnepamäti.

Kedy teda nastal „veľký tresk“ a kto ho spôsobil? Odpoveď znie, že to bol administrátor, keď databázu vytvoril, a to príkazom `CREATE DATABASE` („vytvor databázu“), v ktorom popri názve databázy okrem iného určil aj kódovú stránku, čo je v jazykoch obdarených diakritikou veľmi dôležité (doplňme, že pre slovenčinu sa odporúča hodnota `1250`). Po čase, keď svet databázy prestane byť zaujímavý, nastane jeho „veľký krach“ – administrátor ju zruší, a to príkazom `DROP DATABASE` („strať databázu“). Pravdaže, v súčasnej dobe už na tieto úkony existujú pohodlné grafické prostriedky, takže samotné príkazy sa dejú v pozadí, utajené dokonca pred samotným administrátorom.

3.6.2 Prístupové práva

Tak ako bol správou sveta okolo seba poverený Adam (v preklade človek), aj administrátor môže delegovať niektoré svoje právomoci ostatným v systéme zaregistrovaným používateľom, a to ako jednotlivcom, tak celým skupinám. Aj tu má na to spravidla k dispozícii grafický prostriedok, no, či už s ním, alebo bez neho, vždy ide o vykonanie niektorej z verzií príkazu `GRANT` („poskytni“). Popri samotnom napojení na databázu mu môže povoliť napríklad vytváranie nových tabuliek, čítanie niektorých tabuliek alebo pohľadov, či vkladanie a/alebo mazanie a/alebo modifikáciu dát.

Dôležité je tu vedieť, že právo čítať sa vzťahuje na tabuľku ako celok, nie iba na niektoré jej riadky či stĺpce. Ak taká potreba vyvstane, stačí definovať príslušný pohľad a právo čítať udeliť naň (tak ako sme to spomenuli v stati 3.2.6).

Dodajme ešte, že práva nemusí udeľovať len administrátor. Pri udelení práva s dodatkom `WITH GRANT OPTION` („s možnosťou udeľovať“) môže totiž obdarený používateľ predĺžiť toto právo aj na ďalších používateľov. Podobne pri vytvorení tabuľky má na nej tvorca všetky práva automaticky (a môže ich teda delegovať ďalej).

Ak sa niekto pokúša o príliš veľké sústo zakázaného ovocia zo stromu poznania (databázy), administrátor ho môže z databázového raja vyhnať, a to príkazom (začínajúcim sa) `REVOKE` („odvolaj“), ktorým mu zneužívané privilégiá zruší.

3.6.3 Spojenie s databázou

Ak chce dostatočne privilegovaný používateľ pracovať s databázou, musí sa na ňu pripojiť. Slúži na to príkaz `CONNECT` („pripoj sa“) nasledovaný spojkou `TO` („na“) a názvom databázy. Ak sa autentifikácia používateľa nevykoná automaticky, príkaz treba doplniť slovami `USER` („(ako) používateľ“) s menom používateľa a `USING` („použijúc (heslo)“) s jeho heslom.

Ak už používateľa práca s databázou omrzí, mal by spojenie prerušiť, a to príkazom `CONNECT RESET` („vynuluj pripojenie“) alebo niektorým z príkazov začínajúcich sa slovom `DISCONNECT` („odpoj“).

3.6.4 Zálohovanie

Je všeobecne známe, že používatelia sa delia do dvoch skupín – na tých, ktorí zálohujú, a na tých, ktorí o svoje dáta zatiaľ ešte neprišli... Skôr než nás osud (spolu s našimi dátami) vymaže z tej druhej skupiny, mali by sme radšej dobrovoľne prejsť do tej prvej, a to hneď. Preto (hoci len v krátkosti) uveďme, že v DB2 na zálohovanie slúži príkaz (začínajúci sa slovom) `BACKUP DATABASE` („zálohuj databázu“), v ktorom okrem

mena databázy uvedieme aj miesto, kde sa má záloha uložiť. V prípade nešťastia potom môžeme databázu obnoviť do stavu pri zálohovaní príkazom RESTORE DATABASE („obnov databázu“), ktorý, samozrejme, tiež musí obsahovať informáciu, kde sa záloha obnovovanej databázy nachádza.

3.7 Spolupráca SQL a iných jazykov

3.7.1 Embedded SQL

Naša doterajšia práca s SQL vyzerala tak, že sme písali izolované príkazy, na ktoré systém interaktívne reagoval (či už výpisom tabuliek na obrazovku, alebo oznamom, že nejako modifikoval dáta, prípadne chybovou hláškou). Vzniká však prirodzená otázka, či systém musí komunikovať iba s používateľom, teda či schopnosti SQL nemôžu byť využité nejakým iným systémom. Odpoveďou je existencia tzv. **embedded SQL**, t. j. SQL „vloženého“, „zapusteného“ do iného jazyka.

Všimnime si nasledujúcu ukážku programu v jazyku C++, v ktorom sú vložené kúsky jemne upraveného SQL kódu. Tento hybrid nie je ani čistým C++ kódom, ani SQL príkazom, preto má špeciálnu príponu `.sqc`. Predtým než ho chceme použiť, ho potrebujeme predkompilovať (v DB2 to urobíme pomocou príkazu PREP (skratka pre „prepare“, „priprav“)), čím sa SQL pasáže syntakticky nahradia špeciálnymi príkazmi jazyka C++, a vznikne tak súbor s klasickou príponou `.cpp`. Ten treba naviazať na databázu (príkazom BIND („naviaž“)) a (za použitia knižnice `utilemb`) prekompilovať:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "utilemb.h"

EXEC SQL INCLUDE SQLCA;

int main (char *argv[])
{
  EXEC SQL BEGIN DECLARE SECTION;
    char databaza [8];
    char pouzivatel [8];
    char heslo [8];

    short id;
    char meno [10];
    char priezvisko [10];
  EXEC SQL END DECLARE SECTION;

  strcpy (databaza, argv [1]);
  strcpy (pouzivatel, argv [2]);
  strcpy (heslo, argv [3]);
  EXEC SQL CONNECT TO :databaza USER :pouzivatel USING :heslo;
  EMB_SQL_CHECK("pripojenie na databazu");

  EXEC SQL DECLARE clovek CURSOR FOR
    SELECT id, meno, priezvisko FROM osoba;
  EMB_SQL_CHECK("deklaracia kurzora");

  EXEC SQL OPEN clovek;
  EMB_SQL_CHECK("otvorenie kurzora");

  do
  {
    EXEC SQL FETCH clovek INTO :id, :meno, :priezvisko;
    if (SQLCODE != 0) break;
    printf("Osoba s~cislom %d je %s %s.\n", id, meno, priezvisko);
  } while (1);

  EXEC SQL CLOSE clovek;
  EMB_SQL_CHECK("zatvorenie kurzora");

  EXEC SQL CONNECT RESET;
  EMB_SQL_CHECK("prerusenie pripojenia");

  return 0;
}
```

Pozrime sa teraz bližšie na jednotlivé SQL časti tohto kódu. Spoznáme ich veľmi jednoducho – sú to riadky (resp. oblasti medzi riadkami), ktoré sa začínajú reťazcami `EXEC SQL` (skratka z „execute SQL“, „vykonaj SQL“) alebo `EMB_SQL_CHECK` (skratka z „embedded SQL check“, „kontrola vloženého SQL“).

Príkaz `INCLUDE SQLCA` („zahrň SQLCA“) umožňuje programu komunikovať s databázou, a to prostredníctvom čísel `SQLCODE` („SQL kód“). Úspešné vykonanie príkazu je indikované kódmi `0` alebo `100` (tým druhým v prípade, že príkaz je síce správny, ale stav databázy sa po jeho vykonaní nezmenil, resp., ak to bol dopyt, že odpoveď naň je prázdna). Ostatné kódy indikujú chyby rôzneho druhu.

Časť `DECLARE SECTION` („úsek deklarácií“) ohraničená príslušnými dvoma riadkami obsahuje deklaráciu premenných, ktoré sa (popri normálnom použití v C++ kóde) vyskytujú v SQL pasážach – tam každému ich výskytu predchádza znak `:`. Hneď prvé použitie vidíme v časti `CONNECT`, v ktorej spoznávame príkaz na pripojenie k databáze.

Potom nasleduje dôležitá časť `DECLARE CURSOR FOR` („deklaruj kurzor pre“), kde deklarujeme SQL dopyt, tzv. **kurzor** tým, že za slovo `DECLARE` napíšeme jeho nami zvolené meno (v našom prípade je to `clovek`) a ďalej za slovko `FOR` samotný dopyt (u nás je to `SELECT id, meno, priezvisko FROM osoba`).

Tento príkaz sa vykoná hneď ďalším príkazom `OPEN` („otvor“) (nasledovaným menom kurzora), čím sa vytvorí tabuľka, t. j. množina dát, čakajúcich na ďalšie spracovanie. Kurzor (v súlade s významom tohto slova) si potom môžeme predstaviť ako čosi, čo prebieha po jednotlivých riadkoch tejto tabuľky.

Pekne to vidno v nasledujúcom cykle s príkazom `FETCH INTO` („prines do“) (medzi týmito dvoma slovami je meno kurzora a za nimi toľko premenných, koľko stĺpcov má tabuľka), kde sa aktuálne údaje (t. j. údaje z riadku, na ktorom je práve nastavený kurzor) postupne vkladajú do príslušných premenných (opäť uvádzaných za dvojbodkou) a C++-príkazom `printf` sa vypisujú.

Po skončení cyklu kurzor uzavrieme príkazom `CLOSE` („zatvor“) (za ktorým je meno príslušného kurzora) a spojenie s databázou zabezpečí už známy príkaz `CONNECT RESET`.

Riadky označené `EMB_SQL_CHECK` sú kontrolné body, ktoré v prípade nejakej chyby celú vykonávanú transakciu zrušia. Text v zátvorke potom pomôže identifikovať, na ktorom mieste programu chyba nastala.

Všimnime si, že použitie SQL v tomto príklade bolo **statické** – spôsob spolupráce s databázou bol do detailov dohodnutý už pred spustením programu. Niekedy však parametre SQL príkazov používaných v programe v čase kompilácie ešte nepoznáme, v takom prípade ich musíme spracovať **dynamicky** – až počas behu programu. Nasledujúci príklad sa od predošlého líši tým, že tu chceme vypísať len osoby, ktorých priezvisko sa začína na písmeno dodané ako parameter:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "utilemb.h"

EXEC SQL INCLUDE SQLCA;

int main (char *argv[])
{
  EXEC SQL BEGIN DECLARE SECTION;
    char databaza [8];
    char pouzivatel [8];
    char heslo [8];

    char s[80];
    char pismeno[1];

    short id;
    char meno [10];
    char priezvisko [10];
  EXEC SQL END DECLARE SECTION;

  strcpy (databaza, argv [1]);
  strcpy (pouzivatel, argv [2]);
  strcpy (heslo, argv [3]);
```

```
  EXEC SQL CONNECT TO :databaza USER :pouzivatel USING :heslo;
  EMB_SQL_CHECK("pripojenie na databazu");

  strcpy(s, "SELECT id, meno, priezvisko");
  strcat(s, "FROM osoba");
  strcat(s, "WHERE SUBSTR(priezvisko,1,1) = ?");
  EXEC SQL PREPARE clovek_dopyt FROM :s;
  EMB_SQL_CHECK("priprava (neuplneho) dopytu");

  EXEC SQL DECLARE clovek CURSOR FOR clovek_dopyt;

  strcpy(pismeno, argv [4]);
  EXEC SQL OPEN clovek USING :pismeno;
  EMB_SQL_CHECK("otvorenie kurzora");

  do
  {
    EXEC SQL FETCH clovek INTO :id, :meno, :priezvisko;
    if (SQLCODE != 0) break;
    printf("Osoba s~cislom %d je %s %s.\n", id, meno, priezvisko);
  } while (1);

  EXEC SQL CLOSE clovek;
  EMB_SQL_CHECK("zatvorenie kurzora");

  EXEC SQL CONNECT RESET;
  EMB_SQL_CHECK("prerusenie pripojenia");

  return 0;
}
```

Tu si parametrický SQL dopyt najprv pripravíme pomocou príkazu PREPARE FROM („priprav z“), keď medzi týmito slovami je meno vytváraného dopytu a za nimi je reťazec, z ktorého ho dostaneme a ktorý má na mieste zatiaľ neznámych parametrov znaky ?. Potom preň otvoríme kurzor opäť príkazom OPEN, za ktorým nasleduje slovo USING („pomocou“) a premenné, ktorých hodnoty sa dosadia za otázniky. Zvyšok programu ostáva nezmenený.

3.7.2 Používateľom definované funkcie

V oboch uvedených príkladoch vystupoval jazyk C++ voči SQL ako **hostiteľský**, možná je však spolupráca aj v opačnom smere. V stati 1.2.3 sme spomenuli, že DB2 obsahuje kvantum funkcií všemožného druhu. Môže sa však stať, že používateľa žiadna z nich neuspokojí, a v takom prípade môže použiť funkciu naprogramovanú trebárs v spomínanom jazyku C++. Na prepojenie takejto **používateľom definovanej funkcie** (**user defined function**, **UDF**) s databázou slúži príkaz začínajúci sa CREATE FUNCTION („vytvor funkciu“). V ňom uvedieme ako jej meno v jazyku, kde je pôvodne naprogramovaná, tak jej nové meno, ktoré budeme používať v SQL, samozrejme, spolu s informáciou o dátových typoch jej vstupov a výstupu. A napokon nám **zmysel pre zákony zachovania** nedovolí nespomenúť, že stopy po takejto funkcii zahladíme príkazom začínajúcim sa DROP FUNCTION („strať funkciu“).

4
Teoretické základy databáz

4.1 Formalizácia tabuľky

4.1.1 Formalizácia stĺpca

Po praktických skúsenostiach, ktoré sme získali v predchádzajúcich kapitolách, nastal čas pozrieť sa na databázy exaktnejšie. SQL sme síce v úvode vyhlásili za deklaratívny jazyk, ale ako potenciálni programátori by sme mali mať aspoň hrubú predstavu, ako asi databázový stroj plniaci SQL-príkazy používateľa funguje. Navrhneme preto formálny model, ktorým (čo najobjektovejšie) popíšeme základné pojmy v teórii databáz. Prv než začneme, zaveďme si kvôli jednoduchosti nasledujúce označenie:

Označenie

- Predpokladajme, že máme definované množiny CeléČíslo, DesatinnéČíslo, Reťazec, Dátum, ... s príslušným obsahom. Označme potom DátovýTyp systém obsahujúci (ako prvky) všetky tieto množiny a každý jeho prvok nazveme **dátový typ**.

Táto množina teda zhromažďuje všetky množiny, ktoré prichádzajú do úvahy, tri bodky naznačujú, že sme (možno) nevymenovali všetky možnosti. Teraz už môžeme definovať prvý základný pojem – stĺpec.

Definícia

- **Stĺpcom** nazývame také s, že $s = \langle n, t, p \rangle$ pre nejaké $n \in \text{Reťazec}$, $t \in \text{DátovýTyp}$ a $p \in \{0, 1\}$.
- Reťazec n nazývame **názov** stĺpca s a označujeme ho $s.\text{názov}$.
- Dátový typ t budeme volať **dátový typ** (alebo len **typ**) stĺpca s a značiť ho $s.\text{typ}$.
- Pod **nullabilitou** (alebo v ťažkopádnom preklade **prázdnaschopnosťou**) rozumieme číslo p. Označovať ho budeme $s.\text{nullabilita}$.

Ak teda napríklad $s = \langle \texttt{priezvisko}, \text{Reťazec}, 0 \rangle$, tak s je stĺpec a platí $s.\text{názov} = \texttt{priezvisko}$ (to je názov stĺpca s), $s.\text{typ} = \text{Reťazec}$ (jeho dátový typ) a $s.\text{nullabilita} = 0$ (takže nepripúšťa prázdne hodnoty).

Teraz už môžeme prejsť k definícii centrálneho databázového pojmu – tabuľky. Aby to nebolo príliš jednoduché, poskytneme hneď tri verzie.

4.1.2 Abstraktná tabuľka

Definícia

- Usporiadanú dvojicu $T = \langle \text{M}_T, \text{D}_T \rangle$ budeme nazývať **abstraktná tabuľka**, ak platí:
  - M_T je množina stĺpcov takých, že žiadne dva z nich nemajú rovnaký názov (formálne teda $(\forall s_1, s_2 \in \text{M}_T)(s_1 \neq s_2) \rightarrow (s_1.\text{názov} \neq s_2.\text{názov})$). Túto množinu nazveme **metadáta** tejto abstraktnej tabuľky.
  - D_T je množina funkcií $r : \text{M}_T \rightarrow \bigcup \text{DátovýTyp} \cup \{\texttt{NULL}\}$ takých, že pre každý stĺpec $s \in \text{M}_T$ platí
  $$r(s) \in \begin{cases} s.\text{typ}, & \text{ak } s.\text{nullabilita} = 0, \\ s.\text{typ} \cup \{\texttt{NULL}\}, & \text{ak } s.\text{nullabilita} = 1. \end{cases}$$
  (Vzhľadom na jednoznačnosť názvov stĺpcov budeme namiesto $r(s)$ písať aj $r.(s.\text{názov})$.) Každú takúto funkciu nazývame **záznam** alebo **riadok**, množinu D_T budeme nazývať **dáta** tejto abstraktnej tabuľky.

Objasnime si tieto pojmy na príklade (torza) starej známej databázovej tabuľky `študent` (s dvoma Jánmi Hlúpymi):

meno	priezvisko	pohlavie	dátum_narodenia	ročník	priemer
Ján	Hraško	muž	1987-07-12	1	1,83
Ružena	Šípová	žena	1984-02-01	1	1,22
Ján	Hlúpy	muž	NULL	2	3,00
Ján	Hlúpy	muž	NULL	2	3,00

Táto naša tabuľka je reprezentovaná abstraktnou tabuľkou T, ktorej stĺpce sú $s_1 = \langle \texttt{meno}, \text{Reťazec}, 0\rangle$, $s_2 = \langle \texttt{priezvisko}, \text{Reťazec}, 0\rangle$, $s_3 = \langle \texttt{pohlavie}, \text{Reťazec}, 0\rangle$, $s_4 = \langle \texttt{dátum\_narodenia}, \text{Dátum}, 1\rangle$, $s_5 = \langle \texttt{ročník}, \text{CeléČíslo}, 0\rangle$ a $s_6 = \langle \texttt{priemer}, \text{DesatinnéČíslo}, 1\rangle$. Čo sa týka nullability, uvedomme si, že všade, kde je hodnota 0, prichádza do úvahy aj hodnota 1, možno sa prázdne hodnoty iba zatiaľ – v aktuálnych dátach – neprejavili (z SQL-definície to môžeme zistiť podľa toho, či je stĺpec definovaný ako `NOT NULL`, alebo nie). Naproti tomu nullabilita stĺpca s_4 iná než 1 byť nemôže, pretože tento stĺpec obsahuje aj prázdnu hodnotu. Dôvody, pre ktoré sme stĺpcu s_6 s názvom `priemer` priradili nullabilitu 1, sme už spomínali. Metadáta sú potom množina $\text{M}_T = \{s_1, s_2, s_3, s_4, s_5, s_6\}$. Definícii vyhovujú, lebo všetky názvy stĺpcov sú rôzne. Riadok tabuľky, ktorý je napísaný ako posledný, je funkcia (teda množina usporiadaných dvojíc s rôznymi prvými zložkami)

$$r^4 = \{\langle s_1, \texttt{Ján}\rangle, \langle s_2, \texttt{Hlúpy}\rangle, \langle s_3, \texttt{muž}\rangle, \langle s_4, \texttt{NULL}\rangle, \langle s_5, 2\rangle, \langle s_6, \texttt{3,00}\rangle\}.$$

Definícii vyhovuje, lebo:

- $r^4.\texttt{meno} = r^4(s_1) = \texttt{Ján} \in \text{Reťazec} = s_1.\text{typ}$,
- $r^4.\texttt{priezvisko} = r^4(s_2) = \texttt{Hlúpy} \in \text{Reťazec} = s_2.\text{typ}$,
- $r^4.\texttt{pohlavie} = r^4(s_3) = \texttt{muž} \in \text{Reťazec} = s_3.\text{typ}$,
- $r^4.\texttt{dátum\_narodenia} = r^4(s_4) = \texttt{NULL} \in \text{Dátum} \cup \{\texttt{NULL}\} = s_4.\text{typ} \cup \{\texttt{NULL}\}$ (v tomto prípade je totiž $s_4.\text{nullabilita} = 1$),
- $r^4.\texttt{ročník} = r^4(s_5) = 2 \in \text{CeléČíslo} = s_5.\text{typ}$,
- $r^4.\texttt{priemer} = r^4(s_6) = \texttt{3,00} \in \text{DesatinnéČíslo} \cup \{\texttt{NULL}\} = s_6.\text{typ} \cup \{\texttt{NULL}\}$ (keďže $s_6.\text{nullabilita} = 1$).

Podobne je to aj s ostatnými riadkami tabuľky:

$$r^1 = \{\langle s_1, \texttt{Ján}\rangle, \langle s_2, \texttt{Hraško}\rangle, \langle s_3, \texttt{muž}\rangle, \langle s_4, \texttt{12.7.1987}\rangle, \langle s_5, 1\rangle, \langle s_6, \texttt{1,83}\rangle\},$$
$$r^2 = \{\langle s_1, \texttt{Ružena}\rangle, \langle s_2, \texttt{Šípová}\rangle, \langle s_3, \texttt{žena}\rangle, \langle s_4, \texttt{1.2.1984}\rangle, \langle s_5, 1\rangle, \langle s_6, \texttt{1,22}\rangle\},$$
$$r^3 = \{\langle s_1, \texttt{Ján}\rangle, \langle s_2, \texttt{Hlúpy}\rangle, \langle s_3, \texttt{muž}\rangle, \langle s_4, \texttt{NULL}\rangle, \langle s_5, 2\rangle, \langle s_6, \texttt{3,00}\rangle\}.$$

Všimnime si, že funkcia r^3 je totožná s r^4, dáta tejto (abstraktnej) tabuľky sú teda trojprvková množina $\text{D}_T = \{r^1, r^2, r^3\}$.

4.1.3 Typovaná relácia

Vzhľadom na problémy s implementáciou pojmu množina do hry prichádza aj poradie jej prvkov. Sústreďme sa najprv na stĺpce. Predchádzajúcu definíciu abstraktnej tabuľky preto upravíme – doplníme ju zmienkou o ich poradí. Metadáta i jednotlivé riadky potom nebudeme chápať ako funkcie, ale usporiadané tice.

Definícia

- $T = \langle \text{m}_T, \text{D}_T\rangle$ budeme nazývať **typovaná relácia**, ak platí:
  - m_T je usporiadaná tica stĺpcov $\langle s_1, \ldots, s_n\rangle$. Túto ticu nazveme **metadáta** tejto typovanej relácie.
  - D_T je množina usporiadaných tíc $r = \langle r_1, \ldots, r_n\rangle$ takých, že pre každé $i \in \{1, \ldots, n\}$ platí

$$r_i \in \begin{cases} s_i.\text{typ}, & \text{ak } s_i.\text{nullabilita} = 0, \\ s_i.\text{typ} \cup \{\texttt{NULL}\}, & \text{ak } s_i.\text{nullabilita} = 1. \end{cases}$$

> Každú takúto ticu nazývame **záznam** alebo **riadok**, množinu D_T budeme nazývať **dáta** tejto typovanej relácie.

Uvedomme si, že z hľadiska predikátového počtu sú metadáta (možno okrem názvov stĺpcov) vlastne zovšeobecnenou aritou istého predikátu a dáta sú jeho interpretáciou. (Viac o tom v podkapitole 4.2.)

Vzhľadom na to, že pri typovanej relácii máme jednoznačne určené poradie stĺpcov, nemusíme kvôli ich identifikácii vyžadovať rôznosť ich názvov. Obvykle tak však robíme, pretože v SQL sa odvolávame na stĺpce nie ich poradím (ktoré je vlastne pred používateľom mierne zatajované), ale názvami. Formálne túto dodatočnú podmienku môžeme vyjadriť vzťahom $(\forall i_1, i_2 \in \{1, \ldots, n\})(i_1 \neq i_2) \rightarrow (s_{i_1}.\text{názov} \neq s_{i_2}.\text{názov})$.

Ak z predchádzajúceho príkladu prevezmeme označenie stĺpcov, uvažovaná tabuľka `študent` zodpovedá typovanej relácii T s metadátami $\mathrm{m}_T = \langle s_1, s_2, s_3, s_4, s_5, s_6 \rangle$. Nejde už teda o množinu ako pri abstraktnej tabuľke, ale o ticu. Podobne posledný napísaný riadok v tomto prípade už nie je funkcia, ale tica

$$r^4 = \langle r^4_1, r^4_2, r^4_3, r^4_4, r^4_5, r^4_6 \rangle = \langle \texttt{Ján}, \texttt{Hlúpy}, \texttt{muž}, \texttt{NULL}, \texttt{2}, \texttt{3,00} \rangle.$$

Definícii vyhovuje, lebo platí:

- $r^4_1 = \texttt{Ján} \in \text{Reťazec} = s_1.\text{typ}$,
- $r^4_2 = \texttt{Hlúpy} \in \text{Reťazec} = s_2.\text{typ}$,
- $r^4_3 = \texttt{muž} \in \text{Reťazec} = s_3.\text{typ}$,
- $r^4_4 = \texttt{NULL} \in \text{Dátum} \cup \{\texttt{NULL}\} = s_4.\text{typ} \cup \{\texttt{NULL}\}$,
- $r^4_5 = \texttt{2} \in \text{CeléČíslo} = s_5.\text{typ}$,
- $r^4_6 = \texttt{3,00} \in \text{DesatinnéČíslo} \cup \{\texttt{NULL}\} = s_6.\text{typ} \cup \{\texttt{NULL}\}$.

Podobne je to aj s ostatnými riadkami tabuľky:

$$\begin{aligned} r^1 &= \langle \texttt{Ján}, \texttt{Hraško}, \texttt{muž}, \texttt{12.7.1987}, \texttt{1}, \texttt{1,83} \rangle, \\ r^2 &= \langle \texttt{Ružena}, \texttt{Šípová}, \texttt{žena}, \texttt{1.2.1984}, \texttt{1}, \texttt{1,22} \rangle, \\ r^3 &= \langle \texttt{Ján}, \texttt{Hlúpy}, \texttt{muž}, \texttt{NULL}, \texttt{2}, \texttt{3,00} \rangle. \end{aligned}$$

Aj v tomto prípade platí $r^3 = r^4$ (i keď tentoraz je to rovnosť tíc), dáta tejto typovanej relácie sú preto $\mathrm{D}_T = \{r^1, r^2, r^3\}$.

4.1.4 Reálna tabuľka

Ak vezmeme do úvahy aj poradie riadkov v typovanej relácii, dostávame tabuľku, tak ako ju poznáme:

Definícia

- $T = \langle \mathrm{m}_T, \mathrm{d}_T \rangle$ nazývame **reálna tabuľka** (m_T potom budú **metadáta** a d_T **dáta** tejto reálnej tabuľky), ak existujú $r^1, \ldots, r^k$ také, že:
  - $\mathrm{d}_T = \langle r^1, \ldots, r^k \rangle$,
  - $\langle \mathrm{m}_T, \{r^1, \ldots, r^k\} \rangle$ je typovaná relácia.

Náš príklad `študent` tak možno považovať aj za reálnu tabuľku, ktorej metadáta sú totožné s metadátami príslušnej typovanej relácie, ale dáta sú usporiadaná štvorica $\langle r^1, r^2, r^3, r^4 \rangle$ – naďalej síce platí $r^3 = r^4$, ale tentoraz táto duplicita nie je eliminovaná.

4.1.5 Zhrnutie

Zhrňme to: Každú databázovú tabuľku môžeme reprezentovať troma spôsobmi podľa toho, do akej miery nám záleží na poradí jej stĺpcov či riadkov:

- **abstraktná tabuľka** (nezáleží na poradí stĺpcov ani riadkov),
- **typovaná relácia** (záleží iba na poradí stĺpcov, na poradí riadkov nie),
- **reálna tabuľka** (záleží aj na poradí stĺpcov aj na poradí riadkov).

V ďalších statiach sa sústredíme na kompromisnú druhú možnosť. (Kvôli úplnosti dodajme, že možnosť, pri ktorej záleží iba na poradí riadkov, ale na poradí stĺpcov nie, sa nejaví zmysluplná.)

4.2 Databázová logika

4.2.1 Typované funkcie

V tejto stati upravíme základné definície z logiky predikátového počtu tak, aby lepšie odrážali činnosť databáz. Prv než sa budeme zaoberať termami a podmienkami, budeme potrebovať takzvané **typované funkcie**:

Definícia

- (Zovšeobecnenou) **aritou** funkcie f s n argumentmi nazývame usporiadanú dvojicu $\langle\langle A_1, \ldots, A_n\rangle, V\rangle$, kde A_i je dátový typ i. argumentu a V dátový typ výsledku, t. j. ak $f(x_1, \ldots, x_n) = y$, tak pre každé $i \in \{1, \ldots, n\}$ platí $x_i \in A_i \cup \{\texttt{NULL}\}$ a $y \in V \cup \{\texttt{NULL}\}$.

Napríklad funkcia `YEAR` z SQL (presnejšie, jedna z jej verzií) má aritu $\langle\langle\text{Dátum}\rangle, \text{CeléČíslo}\rangle$, lebo jej jediný argument je dátum (alebo prázdna hodnota) a výsledok je celé číslo (alebo `NULL`). Funkcia `CONCAT` má aritu $\langle\langle\text{Reťazec}, \text{Reťazec}\rangle, \text{Reťazec}\rangle$, lebo oba jej argumenty i výsledok sú reťazce.

Mali by sme sa tiež vyjadriť k postoju funkcií k prázdnym hodnotám. Spravidla platí, že ak má niektorý zo vstupov hodnotu `NULL`, aj výstupná hodnota je prázdna. Výnimočne sa však vyskytujú aj funkcie, ktoré toto pravidlo porušujú, napríklad funkcii `VALUE` stačí na neprázdny výsledok jediná neprázdna hodnota. Cieľom nasledujúcej definície je podchytiť toto správanie:

Definícia

- Nech f je funkcia s n argumentmi. Nech $p_1, \ldots, p_n \in \{0, 1\}$ a tiež $q \in \{0, 1\}$. Ak z toho, že pre ľubovoľné $i \in \{1, \ldots, n\}$ platí
$$x_i \begin{cases} = \texttt{NULL}, & \text{ak } p_i = 1, \\ \neq \texttt{NULL}, & \text{ak } p_i = 0, \end{cases}$$
vyplýva, že
$$f(x_1, \ldots, x_n) \begin{cases} = \texttt{NULL}, & \text{ak } q = 1, \\ \neq \texttt{NULL}, & \text{ak } q = 0, \end{cases}$$
tak usporiadanú dvojicu $\langle\langle p_1, \ldots, p_n\rangle, q\rangle$ nazveme **nullabilitným pravidlom** funkcie f.

Funkcia `CONCAT` má nullabilitné pravidlo $\langle\langle 0, 0\rangle, 0\rangle$, lebo ak sú oba argumenty rôzne od `NULL`, aj výsledok bude taký. Ak je však hociktorý z nich rovný `NULL`, prázdna hodnota bude aj vo výsledku, preto ďalšie nullabilitné pravidlá funkcie `CONCAT` sú $\langle\langle 0, 1\rangle, 1\rangle$, $\langle\langle 1, 0\rangle, 1\rangle$ a $\langle\langle 1, 1\rangle, 1\rangle$. Zhrnutím týchto štyroch pravidiel dostávame vlastne postoj tejto funkcie k prázdnym hodnotám:

Definícia

- **Nullabilitným správaním** funkcie f nazývame množinu všetkých jej nullabilitných pravidiel.

Nullabilitné správanie funkcie `CONCAT` je teda $\{\langle\langle 0, 0\rangle, 0\rangle, \langle\langle 0, 1\rangle, 1\rangle, \langle\langle 1, 0\rangle, 1\rangle, \langle\langle 1, 1\rangle, 1\rangle\}$. Zato pre funkciu `VALUE` (presnejšie pre jej binárne verzie) je to $\{\langle\langle 0, 0\rangle, 0\rangle, \langle\langle 1, 0\rangle, 0\rangle, \langle\langle 0, 1\rangle, 0\rangle, \langle\langle 1, 1\rangle, 1\rangle\}$ (keďže na neprázdnosť výsledku treba (a stačí) neprázdnosť ktoréhokoľvek argumentu).

Uvedomme si ešte, že nullabilitné správanie je funkcia – nemôžu existovať dve rôzne pravidlá s rovnakou prvou zložkou. Nullabilitné správanie n-árnej funkcie preto nemá viac ako 2^n pravidiel.

4.2.2 Termy

Teraz už môžeme definovať term:

Definícia

- Nech T je typovaná relácia, ktorej všetky stĺpce majú rôzne názvy. Potom **T-termy** (resp. skrátene len **termy**) sú definované ako takáto induktívna štruktúra (a to včítane ich **typu** a **nullability**):

  1a Všetky prvky z množiny $\bigcup \mathrm{DátovýTyp} \cup \{\texttt{NULL}\}$ sú T-termy, nazývané **konštanty**.
  - Typom každej neprázdnej konštanty bude tá (najmenšia) množina z $\mathrm{DátovýTyp}$, v ktorej sa nachádza, a jej nullabilita bude 0.
  - Typ konštanty `NULL` nie je definovaný a jej nullabilita je 1.

  1b Ak s je stĺpec T, tak $s.\mathrm{názov}$ je T-term, zvaný **T-premenná** (alebo len **premenná**). Jeho typ bude $s.\mathrm{typ}$ a nullabilita zas $s.\mathrm{nullabilita}$.

  2 Nech f je funkcia s aritou $\langle\langle A_1, \ldots, A_n\rangle, V\rangle$, nech $t_1, \ldots, t_n$ sú T-termy a nech pre každé $i \in \{1, \ldots, n\}$ je $A_i = t_i.\mathrm{typ}$ a $p_i = t_i.\mathrm{nullabilita}$. Potom $f(t_1, \ldots, t_n)$ je tiež T-term. Jeho typ bude V a jeho nullabilita:
  - 0, ak má funkcia f nullabilitné pravidlo $\langle\langle p_1, \ldots, p_n\rangle, 0\rangle$,
  - 1 inak.

  (Dodajme, že na rozlíšenie konštánt typu $\mathrm{Reťazec}$ od premenných ich budeme niekedy dávať do apostrofov.)

Napríklad pre našu typovanú reláciu T, ktorú teraz zapíšeme takýmto spôsobom (v prvom riadku záhlavia sú názvy stĺpcov, v druhom ich dátové typy a v treťom nullability):

`meno`	`priezvisko`	`pohlavie`	`dátum_narodenia`	`ročník`	`priemer`
Reťazec	Reťazec	Reťazec	Dátum	CeléČíslo	DesatinnéČíslo
0	0	0	1	0	1
`Ján`	`Hraško`	`muž`	`12.7.1987`	`1`	`1,83`
`Ružena`	`Šípová`	`žena`	`1.2.1984`	`1`	`1,22`
`Ján`	`Hlúpy`	`muž`	`NULL`	`2`	`3,00`

je jeden z T-termov `YEAR(dátum_narodenia)+4` s typom CeléČíslo a nullabilitou 1. Môžeme totiž napísať jeho vytvárajúcu postupnosť:

$t_0 =$ `dátum_narodenia` (podľa bodu 1b),
  typ Dátum (je to dátový typ stĺpca s názvom `dátum_narodenia`),
  nullabilita 1 (je to nullabilita tohto stĺpca),
$t_1 =$ `YEAR(dátum_narodenia)` $=$ `YEAR(`t_0`)` (podľa bodu 2),
  typ CeléČíslo (lebo arita funkcie `YEAR` je $\langle\langle\mathrm{Dátum}\rangle, \mathrm{CeléČíslo}\rangle$),
  nullabilita 1 (lebo $\langle\langle 1\rangle, 0\rangle$ nie je nullabilitné pravidlo funkcie `YEAR`),
$t_2 =$ `4` (podľa bodu 1a),
  typ CeléČíslo (lebo `4` $\in$ CeléČíslo),
  nullabilita 0 (lebo `4` $\neq$ `NULL`),
$t_3 =$ `YEAR(dátum_narodenia)+4` $= t_1$`+`t_2 (podľa bodu 2),
  typ CeléČíslo (lebo arita funkcie `+` je (v tomto prípade) $\langle\langle\mathrm{CeléČíslo}, \mathrm{CeléČíslo}\rangle, \mathrm{CeléČíslo}\rangle$),
  nullabilita 1 (lebo $\langle\langle 1, 0\rangle, 0\rangle$ nie je nullabilitné pravidlo funkcie `+`).

Uvedomme si, že z hľadiska matematickej logiky sme sa dopustili istej nepresnosti, keď sme stotožnili symboly konštánt i funkcií s ich interpretáciami. Hodnota T-termu pre nejaký riadok preto bude vyzerať takto (všimnime si, že tu vyžadujeme rôznosť mien stĺpcov):

Definícia

- Nech T je typovaná relácia s metadátami $\mathrm{m}_T = \langle s_1, \ldots, s_n \rangle$, pričom všetky jej stĺpce majú rôzne názvy, a nech $r = \langle r_1, \ldots, r_n \rangle$ je jej riadok. Potom definujme indukciou funkciu h_r nazývanú **hodnota termu** v riadku r takto:

  1a Hodnota ľubovoľnej konštanty c je ona sama, t. j. $\mathrm{h}_r(c) = c$.

  1b Hodnota T-premennej $s_i.\mathrm{názov}$ je r_i, t. j. $\mathrm{h}_r(s_i.\mathrm{názov}) = r_i$.

  2 Ak $t = f(t_1, \ldots, t_n)$, tak $\mathrm{h}_r(t) = \underline{f}(\mathrm{h}_r(t_1), \ldots, \mathrm{h}_r(t_n))$, kde $\underline{f}$ je funkcia s menom f.

Napríklad hodnota nášho T-termu `YEAR(dátum_narodenia)+4` v hociktorom riadku $r = \langle r_1, r_2, r_3, r_4, r_5, r_6 \rangle$ je potom

$$\begin{aligned}
&\mathrm{h}_r(\texttt{YEAR(dátum_narodenia)+4}) = \\
&= \mathrm{h}_r(\texttt{YEAR(dátum_narodenia)}) \,\underline{+}\, \mathrm{h}_r(4) \text{ (podľa bodu 2)}, \\
&= \underline{\texttt{YEAR}}(\mathrm{h}_r(\texttt{dátum_narodenia})) \,\underline{+}\, \mathrm{h}_r(4) \text{ (podľa bodu 2)}, \\
&= \underline{\texttt{YEAR}}(\mathrm{h}_r(\texttt{dátum_narodenia})) \,\underline{+}\, 4 \text{ (podľa bodu 1a)}, \\
&= \underline{\texttt{YEAR}}(r_4) \,\underline{+}\, 4 \text{ (podľa bodu 1b)},
\end{aligned}$$

takže pre riadok r^1 Jána Hraška to bude

$$\mathrm{h}_{r^1}(\texttt{YEAR(dátum\_narodenia)+4}) = \underline{\texttt{YEAR}}(r^1_4) \,\underline{+}\, 4 = \underline{\texttt{YEAR}}(\texttt{12.7.1987}) \,\underline{+}\, 4 = \texttt{1987} \,\underline{+}\, 4 = \texttt{1991},$$

pre riadok r^2 Ruženy Šípovej

$$\mathrm{h}_{r^2}(\texttt{YEAR(dátum\_narodenia)+4}) = \underline{\texttt{YEAR}}(r^2_4) \,\underline{+}\, 4 = \underline{\texttt{YEAR}}(\texttt{1.2.1984}) \,\underline{+}\, 4 = \texttt{1984} \,\underline{+}\, 4 = \texttt{1988}$$

a napokon pre riadok r^3 Jána Hlúpeho

$$\mathrm{h}_{r^3}(\texttt{YEAR(dátum\_narodenia)+4}) = \underline{\texttt{YEAR}}(r^3_4) \,\underline{+}\, 4 = \underline{\texttt{YEAR}}(\texttt{NULL}) \,\underline{+}\, 4 = \texttt{NULL} \,\underline{+}\, 4 = \texttt{NULL}.$$

4.2.3 Podmienky

Teraz upresníme pojem logickej podmienky:

Definícia

- Nech T je typovaná relácia so stĺpcami rôznych názvov. **Elementárna T-podmienka** bude ľubovoľný zápis v jednom z týchto tvarov:

  1 $t_1 \,@\, t_2$, kde t_1 a t_2 sú T-termy rovnakého typu a $@$ je jedno z porovnaní `=`, `>`, `<`, `>=`, `<=` a `!=` (resp. alternatívne `<>`). (Úprimne povedané, v praxi sa pripúšťa aj prípad nerovnakých typov, musia však byť aspoň kompatibilné. V takom prípade však môžeme uvažovať o nenapísanej (čiže implicitnej) funkcii, ktorá zabezpečí potrebnú konverziu, a preto možno pri všetkej počestnosti ostať pri podmienke rovnakosti typov.)

  2 t `IS NULL`, kde t je T-term.

Táto definícia si nerobí nároky na úplnosť – mohla by byť doplnená o ďalšie typy elementárnych T-podmienok používajúcich napríklad `CASE`, `IN`, `BETWEEN`, vnútorné dopyty, atď.. Kvôli zjednodušeniu situácie sa však nimi nebudeme zaoberať (dobrá výhovorka stojí groš).

Definícia

- **T-podmienka** (resp. kratšie **podmienka**) bude ľubovoľný výrok, ktorého elementárne podvýroky sú iba elementárne T-podmienky a ktorý nemá iné spojky ako negáciu (`NOT`), konjunkciu (`AND`) a disjunkciu (`OR`).

Pre našu tabuľku `študent` je T-podmienkou napríklad

$$\texttt{(SUBSTR(meno,1,1)='J') AND (NOT (dátum\_narodenia IS NULL))},$$

jej elementárne podvýroky sú `SUBSTR(meno,1,1)='J'` a `dátum_narodenia IS NULL`.

Treba sa ešte naučiť takéto podmienky vyhodnocovať v logike s troma hodnotami 0, 1 a $?$ (čo sú formálnejšie verzie hodnôt *nepravda*, *pravda* a *neznáme* zo state 1.3.1):

Definícia

- Nech T je typovaná relácia. Definujme funkciu v_r **vyhodnotenia T-podmienky** φ v riadku r takto:

  1.1 Ak $\varphi = t_1 \,@\, t_2$, tak platí:

  $$\mathrm{v}_r(\varphi) = \begin{cases} ?, & \text{ak } \mathrm{h}_r(t_1) = \texttt{NULL} \text{ alebo } \mathrm{h}_r(t_2) = \texttt{NULL}, \\ 1, & \text{ak } \mathrm{h}_r(t_1), \mathrm{h}_r(t_2) \neq \texttt{NULL} \text{ a platí } \mathrm{h}_r(t_1) \,\underline{@}\, \mathrm{h}_r(t_2), \\ 0, & \text{ak } \mathrm{h}_r(t_1), \mathrm{h}_r(t_2) \neq \texttt{NULL} \text{ a neplatí } \mathrm{h}_r(t_1) \,\underline{@}\, \mathrm{h}_r(t_2), \end{cases}$$

  kde $\underline{@}$ je porovnanie so značkou $@$.

  1.2 Ak $\varphi = t$ `IS NULL`, tak platí:

  $$\mathrm{v}_r(\varphi) = \begin{cases} 1, & \text{ak } \mathrm{h}_r(t) = \texttt{NULL}, \\ 0, & \text{ak } \mathrm{h}_r(t) \neq \texttt{NULL}. \end{cases}$$

  (Všimnime si, že tu hodnota $?$ nastať nemôže.)

  2a Ak $\varphi = \texttt{NOT}\ \psi$, tak je hodnota funkcie v_r daná tabuľkou:

$\mathrm{v}_r(\psi)$	$\mathrm{v}_r(\texttt{NOT}\ \psi)$
0	1
?	?
1	0

  2b Ak $\varphi = \psi_1\ \texttt{AND}\ \psi_2$ alebo $\varphi = \psi_1\ \texttt{OR}\ \psi_2$, tak je hodnota funkcie v_r daná tabuľkou:

$\mathrm{v}_r(\psi_1)$	$\mathrm{v}_r(\psi_2)$	$\mathrm{v}_r(\psi_1\ \texttt{AND}\ \psi_2)$	$\mathrm{v}_r(\psi_1\ \texttt{OR}\ \psi_2)$
0	0	0	0
0	?	0	?
0	1	0	1
?	0	0	?
?	?	?	?
?	1	?	1
1	0	0	1
1	?	?	1
1	1	1	1

Všimnime si, že ak budeme pod znakom $?$ rozumieť $\frac{1}{2}$, spojky zodpovedajú booleovským funkciám definovaným vzťahmi $\mathrm{B}_{\texttt{NOT}}(x) = 1 - x$, $\mathrm{B}_{\texttt{AND}}(x, y) = \min\{x, y\}$ a $\mathrm{B}_{\texttt{OR}}(x, y) = \max\{x, y\}$.

Vyhodnoťme našu T-podmienku `(SUBSTR(meno,1,1)='J') AND (NOT (dátum_narodenia IS NULL))` vo všetkých troch riadkoch tabuľky `študent`:

- Keďže pre riadok r^1 Jána Hraška platí $\mathrm{h}_{r^1}(\texttt{SUBSTR(meno,1,1)}) = \texttt{J}$ a $\mathrm{h}_{r^1}(\texttt{dátum\_narodenia}) = \texttt{12.7.1987} \neq \texttt{NULL}$, máme

$$\begin{aligned}
&\mathrm{v}_{r^1}(\texttt{(SUBSTR(meno,1,1)='J') AND (NOT (dátum_narodenia IS NULL))}) = \\
&= \min\{\mathrm{v}_{r^1}(\texttt{SUBSTR(meno,1,1)='J'}), \mathrm{v}_{r^1}(\texttt{NOT (dátum_narodenia IS NULL)})\}, \\
&= \min\{\mathrm{v}_{r^1}(\texttt{SUBSTR(meno,1,1)='J'}), 1 - \mathrm{v}_{r^1}(\texttt{dátum_narodenia IS NULL})\}, \\
&= \min\{1, 1-0\}, \\
&= 1.
\end{aligned}$$

- Pre riadok r^2 Ruženy Šípovej je $\mathrm{h}_{r^2}(\texttt{SUBSTR(meno,1,1)}) = \texttt{R} \neq \texttt{J}$ a $\mathrm{h}_{r^2}(\texttt{dátum\_narodenia}) = \texttt{1.2.1984} \neq \texttt{NULL}$, preto

$$\begin{aligned}
&\mathrm{v}_{r^2}(\texttt{(SUBSTR(meno,1,1)='J') AND (NOT (dátum_narodenia IS NULL))}) = \\
&= \min\{\mathrm{v}_{r^2}(\texttt{SUBSTR(meno,1,1)='J'}), \mathrm{v}_{r^2}(\texttt{NOT (dátum_narodenia IS NULL)})\}, \\
&= \min\{\mathrm{v}_{r^2}(\texttt{SUBSTR(meno,1,1)='J'}), 1 - \mathrm{v}_{r^2}(\texttt{dátum_narodenia IS NULL})\}, \\
&= \min\{0, 1-0\}, \\
&= 0.
\end{aligned}$$

- Napokon pre riadok r^3 Jána Hlúpeho je $\mathrm{h}_{r^3}(\texttt{SUBSTR(meno,1,1)}) = \texttt{J}$ a $\mathrm{h}_{r^3}(\texttt{dátum\_narodenia}) = \texttt{NULL}$, takže

$$\begin{aligned}
&\mathrm{v}_{r^3}(\texttt{(SUBSTR(meno,1,1)='J') AND (NOT (dátum_narodenia IS NULL))}) = \\
&= \min\{\mathrm{v}_{r^3}(\texttt{SUBSTR(meno,1,1)='J'}), \mathrm{v}_{r^3}(\texttt{NOT (dátum_narodenia IS NULL)})\}, \\
&= \min\{\mathrm{v}_{r^3}(\texttt{SUBSTR(meno,1,1)='J'}), 1 - \mathrm{v}_{r^3}(\texttt{dátum_narodenia IS NULL})\}, \\
&= \min\{1, 1-1\}, \\
&= 0.
\end{aligned}$$

Podmienka `(SUBSTR(meno,1,1)='J') AND (NOT (dátum_narodenia IS NULL))` je teda splnená iba v riadku r^1.

4.3 Relačná algebra

Už sme povedali, že tabuľku môžeme chápať ako (typovanú) reláciu. Ukážeme, že v reči takýchto relácií vieme vyjadriť aj základné dopyty v SQL.

4.3.1 Transformácia stĺpcov, projekcia a premenovanie stĺpcov

Začneme základným príkazom `SELECT` (`DISTINCT`) t_1 (`AS` a_1), ..., t_k (`AS` a_k) `FROM` T, kde T zodpovedá príslušnej typovanej relácii, $t_1, \ldots, t_k$ sú T-termy a (prípadné) $a_1, \ldots, a_k$ aliasy. (`DISTINCT` tam musí byť preto, lebo pri typovaných reláciách ignorujeme prípadnú duplicitu riadkov.)

Definícia

- **Transformáciou stĺpcov** nazývame operátor τ, ktorého prvým argumentom je typovaná relácia T so stĺpcami rôznych názvov a druhým usporiadaná k-tica párov $p = \langle\langle t_1, a_1\rangle, \ldots, \langle t_k, a_k\rangle\rangle$, kde každé t_i je T-term (syntakticky nerovný `NULL`) a každé a_i reťazec znakov. Jej výsledkom je typovaná relácia $R = \tau(T, p)$, ktorej metadáta sú

$$\mathrm{m}_R = \langle\langle a_1, t_1.\mathrm{typ}, t_1.\mathrm{nullabilita}\rangle, \ldots, \langle a_k, t_k.\mathrm{typ}, t_k.\mathrm{nullabilita}\rangle\rangle$$

  a dáta

$$\mathrm{D}_R = \{\langle h_r(t_1), \ldots, h_r(t_k)\rangle : r \in \mathrm{D}_T\}.$$

Napríklad pre známu typovanú reláciu T zodpovedajúcu tabuľke `student`:

meno	priezvisko	pohlavie	dátum_narodenia	ročník	priemer
Reťazec	Reťazec	Reťazec	Dátum	CeléČíslo	DesatinnéČíslo
0	0	0	1	0	1
Ján	Hraško	muž	12.7.1987	1	1,83
Ružena	Šípová	žena	1.2.1984	1	1,22
Ján	Hlúpy	muž	NULL	2	3,00

je výsledok tohto operátora zodpovedajúci dopytu:

```
SELECT DISTINCT
  priezvisko,
  YEAR(dátum_narodenia)+4 AS rok
FROM student
```

typovaná relácia $R = \tau(T, \langle\langle \texttt{priezvisko}, \texttt{priezvisko}\rangle, \langle \texttt{YEAR(dátum\_narodenia)+4}, \texttt{rok}\rangle\rangle)$ s metadátami

$$\mathrm{m}_R = \langle\langle \texttt{priezvisko}, \mathrm{Reťazec}, 0\rangle, \langle \texttt{rok}, \mathrm{CeléČíslo}, 1\rangle\rangle$$

a dátami

$$\mathrm{D}_R = \{\langle \texttt{Hraško}, 1991\rangle, \langle \texttt{Šípová}, 1988\rangle, \langle \texttt{Hlúpy}, \texttt{NULL}\rangle\},$$

teda v grafickej podobe:

priezvisko	rok
Reťazec	CeléČíslo
0	1
Hraško	1991
Šípová	1988
Hlúpy	NULL

Poznamenajme, že ak niektorý alias a_i nie je explicitne uvedený, názov príslušného stĺpca novej typovanej relácie je definovaný iba v prípade, že $t_i = s.\text{názov}$ pre nejaký stĺpec $s \in \text{m}_T$, a potom platí $a_i = t_i = s.\text{názov}$.

Uvedomme si, že špeciálnym prípadom transformácie je pomerne známa databázová operácia **projekcia**, keď pre každé $i \in \{1, \ldots, k\}$ je t_i názvom niektorého stĺpca s_{j_i} a $a_i = t_i$. Projekciu potom zjednodušene zapisujeme

$$\pi(T, \langle j_1, \ldots, j_k \rangle) = \tau(T, \langle \langle t_1, t_1 \rangle, \ldots, \langle t_k, t_k \rangle \rangle).$$

Napríklad pre našu reláciu T projekcia $\pi(T, \langle 6, 2 \rangle)$ zodpovedajúca dopytu:

```
SELECT DISTINCT
  priemer,
  priezvisko
FROM študent
```

(vyberáme teda 6. a 2. stĺpec, a to v takomto poradí) dáva:

priemer	priezvisko
DesatinnéČíslo	Reťazec
1	0
1,83	Hraško
1,22	Šípová
3,00	Hlúpy

Stojí za povšimnutie, že nullabilita stĺpca `priemer` je 1, aj keď žiadna hodnota v ňom nie je prázdna. Je to preto, že sme sa pre takúto nullabilitu rozhodli už skôr – v diskusii o metadátach tabuľky – a projekcia (podľa svojej definície) nullability stĺpcov preberaných z pôvodnej tabuľky zachováva.

Ďalším špeciálnym prípadom transformácie je **premenovanie stĺpcov**, keď sú vybraté všetky stĺpce v pôvodnom poradí, teda keď $\langle t_1, \ldots, t_k \rangle = \langle s_1.\text{názov}, \ldots, s_n.\text{názov} \rangle$, pričom $\langle s_1, \ldots, s_n \rangle = \text{m}_T$ (z čoho tiež $k = n$). Ak $\{j_1, \ldots, j_m\}$ je množina práve tých indexov j z $\{1, \ldots, n\}$, pre ktoré $a_j \neq t_j$, premenovanie zapíšeme zjednodušene

$$\varrho(T, \{\langle j_1, a_{j_1} \rangle, \ldots, \langle j_m, a_{j_m} \rangle\}) = \tau(T, \langle \langle s_1.\text{názov}, a_1 \rangle, \ldots, \langle s_n.\text{názov}, a_n \rangle \rangle).$$

Pre našu reláciu T premenovanie $\varrho(T, \{\langle 5, \texttt{rok\_štúdia} \rangle, \langle 6, \texttt{prospech} \rangle\})$ zodpovedajúce dopytu:

```
SELECT DISTINCT
  meno,
  priezvisko,
  pohlavie,
  dátum_narodenia,
  ročník AS rok_štúdia,
  priemer AS prospech
FROM študent
```

dáva:

meno	priezvisko	pohlavie	dátum_narodenia	rok_štúdia	prospech
Reťazec	Reťazec	Reťazec	Dátum	CeléČíslo	DesatinnéČíslo
0	0	0	1	0	1
Ján	Hraško	muž	12.7.1987	1	1,83
Ružena	Šípová	žena	1.2.1984	1	1,22
Ján	Hlúpy	muž	NULL	2	3,00

4.3.2 Selekcia riadkov

Máme už operáciu o stĺpcoch, teraz sa sústredíme na riadky. Budeme formalizovať dopyt `SELECT * FROM` T `WHERE` φ, kde T zodpovedá príslušnej typovanej relácii a φ je T-podmienka.

Definícia

- **Selekcia riadkov** bude operátor σ, ktorého prvým argumentom je typovaná relácia T a druhým T-podmienka φ. Jej výsledkom je typovaná relácia $R = \sigma(T, \varphi)$, ktorej metadáta sú

$$\mathrm{m}_R = \mathrm{m}_T$$

  a dáta

$$\mathrm{D}_R = \{r \in \mathrm{D}_T : \mathrm{v}_r(\varphi) = 1\}.$$

Pre našu tabuľku T a podmienku `(SUBSTR(meno,1,1)='J') AND (NOT (dátum_narodenia IS NULL))` je výsledok selekcie takýto:

meno	priezvisko	pohlavie	dátum_narodenia	ročník	priemer
Reťazec	Reťazec	Reťazec	Dátum	CeléČíslo	DesatinnéČíslo
0	0	0	1	0	1
Ján	Hraško	muž	12.7.1987	1	1,83

Stĺpce typovanej relácie teda ostali nezmenené a z riadkov sa vybrali len tie, ktoré spĺňajú danú podmienku.

4.3.3 Spojenia dvoch tabuliek, karteziánsky súčin

Vybavili sme operácie na jednej tabuľke, poďme na dvojtabuľkové. Budeme formalizovať dopyty tvaru `SELECT * FROM` T^1 (`LEFT OUTER`, `RIGHT OUTER`, `FULL OUTER`, alebo (aj implicitné) `INNER`) `JOIN` T^2 `ON` $s^1_{j^1_1}.\text{názov} = s^2_{j^2_1}.\text{názov}$ `AND` $\ldots$ `AND` $s^1_{j^1_k}.\text{názov} = s^2_{j^2_k}.\text{názov}$.

Najprv však bude užitočné zadefinovať konkatenáciu tíc (čo nie je nič iné ako známe lepidlo):

Definícia

- **Konkatenácia** k-tice $\langle x_1, \ldots, x_k\rangle$ a n-tice $\langle y_1, \ldots, y_n\rangle$ bude $(k+n)$-tica

$$\langle x_1, \ldots, x_k\rangle \,||\, \langle y_1, \ldots, y_n\rangle = \langle x_1, \ldots, x_k, y_1, \ldots, y_n\rangle.$$

Teraz nám už nič nebráni definovať spojenia:

Definícia

- Nech T^1 a T^2 sú typované relácie, nech ďalej platí $\mathrm{m}_{T^1} = \langle s^1_1, \ldots, s^1_{n^1}\rangle$, $\mathrm{m}_{T^2} = \langle s^2_1, \ldots, s^2_{n^2}\rangle$ a $P = \{\langle j^1_1, j^2_1\rangle, \ldots, \langle j^1_k, j^2_k\rangle\}$, kde $j^1_1, \ldots, j^1_k$ sú indexy niektorých stĺpcov relácie T^1 a $j^2_1, \ldots, j^2_k$ indexy stĺpcov relácie T^2 (t. j. $\{j^1_1, \ldots, j^1_k\} \subseteq \{1, \ldots, n^1\}$ a $\{j^2_1, \ldots, j^2_k\} \subseteq \{1, \ldots, n^2\}$). Potom definujme:
  - **spojenie** (zvané aj **vnútorné spojenie**) $\iota(T^1, T^2, P) = \iota^{\mathrm{J}}(T^1, T^2, P) = J$,
  - **ľavé vonkajšie spojenie** $\iota^{\mathrm{L}}(T^1, T^2, P) = L$,
  - **pravé vonkajšie spojenie** $\iota^{\mathrm{R}}(T^1, T^2, P) = R$,
  - **úplné vonkajšie spojenie** $\iota^{\mathrm{F}}(T^1, T^2, P) = F$,

  kde J, L, R a F sú typované relácie také, že:
  - $\mathrm{m}_J = \mathrm{m}_{T^1} \,||\, \mathrm{m}_{T^2}$,

- $\mathrm{m}_L = \mathrm{m}_{T^1} \,||\, \langle t^2_1, \ldots, t^2_{n^2} \rangle$, pričom pre všetky $i \in \{1, \ldots, n^2\}$ platí:
  - $t^2_i.\text{názov} = s^2_i.\text{názov}$,
  - $t^2_i.\text{typ} = s^2_i.\text{typ}$,
  - $t^2_i.\text{nullabilita} = 1$,
- $\mathrm{m}_R = \langle t^1_1, \ldots, t^1_{n^1} \rangle \,||\, \mathrm{m}_{T^2}$, pričom pre všetky $i \in \{1, \ldots, n^1\}$ platí:
  - $t^1_i.\text{názov} = s^1_i.\text{názov}$,
  - $t^1_i.\text{typ} = s^1_i.\text{typ}$,
  - $t^1_i.\text{nullabilita} = 1$,
- $\mathrm{m}_F = \langle t^1_1, \ldots, t^1_{n^1} \rangle \,||\, \langle t^2_1, \ldots, t^2_{n^2} \rangle$

a:

- $\mathrm{D}_J = \{r^1 \,||\, r^2 : (r^1 \in \mathrm{D}_{T^1}) \wedge (r^2 \in \mathrm{D}_{T^2}) \wedge ((\forall i \in \{1, \ldots, k\})\ r^1_{j^1_i} = r^2_{j^2_i})\}$,
- $\mathrm{D}_L = \mathrm{D}_J \cup \{r^1 \,||\, \langle \underbrace{\texttt{NULL}, \ldots, \texttt{NULL}}_{n^2\times} \rangle : (r^1 \in \mathrm{D}_{T^1}) \wedge (\neg((\exists r^2 \in \mathrm{D}_{T^2})\ r^1 \,||\, r^2 \in \mathrm{D}_J))\}$,
- $\mathrm{D}_R = \mathrm{D}_J \cup \{\langle \underbrace{\texttt{NULL}, \ldots, \texttt{NULL}}_{n^1\times} \rangle \,||\, r^2 : (r^2 \in \mathrm{D}_{T^2}) \wedge (\neg((\exists r^1 \in \mathrm{D}_{T^1})\ r^1 \,||\, r^2 \in \mathrm{D}_J))\}$,
- $\mathrm{D}_F = \mathrm{D}_L \cup \mathrm{D}_R$.

Najlepšie zrejme bude ilustrovať tieto definície na okresanej verzii príkladu zo statí 2.1.2 a 2.1.3, kde sme sa so spojeniami zoznámili. Nech T^1 je typovaná relácia:

meno	priezvisko	id_bydlisko
Reťazec	Reťazec	CeléČíslo
0	0	1
Ján	Hraško	1
Ružena	Šípová	1
Aladár	Baba	2
Ferdinand	Mravec	3
Ján	Polienko	1
Juraj	Truľo	1
Jana	Botková	NULL
Dana	Botková	NULL
Ján	Hlúpy	1
Aladár	Miazga	NULL
Mikuláš	Myšiak	4
Donald	Káčer	4
Jozef	Námorník	NULL
Peter	Pan	5

typovaná relácia T^2 nech vyzerá takto:

id	názov
CeléČíslo	Reťazec
0	0
1	Za siedmimi horami a siedmimi dolami
2	Kalifát Bagdad
3	Mravenisko
4	Hollywood
5	Neverland
6	Haliganda

a množina P nech je tvorená (ako to v drvivej väčšine prípadov býva) jediným prvkom $\langle 3, 1\rangle$. Potom platí:

- Počty stĺpcov sú $n^1 = 3$ a $n^2 = 2$.
- Stĺpce typovanej relácie T^1 sú $s_1^1 = \langle \texttt{meno}, \text{Reťazec}, 0\rangle$, $s_2^1 = \langle \texttt{priezvisko}, \text{Reťazec}, 0\rangle$ a $s_3^1 = \langle \texttt{id\_bydlisko}, \text{CeléČíslo}, 1\rangle$, stĺpce T^2 sú $s_1^2 = \langle \texttt{id}, \text{CeléČíslo}, 0\rangle$, $s_2^2 = \langle \texttt{názov}, \text{Reťazec}, 0\rangle$. Potom $\mathrm{m}_{T^1} = \langle s_1^1, s_2^1, s_3^1\rangle$ a $\mathrm{m}_{T^2} = \langle s_1^2, s_2^2\rangle$.
- Keďže $P = \{\langle 3, 1\rangle\}$, počet dvojíc prepájajúcich stĺpcov je $k = 1$ a ich indexy sú $j_1^1 = 3$ a $j_1^2 = 1$.
- Stĺpce s upravenou nullabilitou sú (pre T^1) $t_1^1 = \langle \texttt{meno}, \text{Reťazec}, 1\rangle$, $t_2^1 = \langle \texttt{priezvisko}, \text{Reťazec}, 1\rangle$ a $t_3^1 = \langle \texttt{id\_bydlisko}, \text{CeléČíslo}, 1\rangle$ a ďalej (pre T^2) $t_1^2 = \langle \texttt{id}, \text{CeléČíslo}, 1\rangle$ a $t_2^2 = \langle \texttt{názov}, \text{Reťazec}, 1\rangle$.
- Riadky v tabuľke T^1 sú $\mathrm{D}_{T^1} = \{\langle \texttt{Ján}, \texttt{Hraško}, 1\rangle, \langle \texttt{Ružena}, \texttt{Šípová}, 1\rangle, \langle \texttt{Aladár}, \texttt{Baba}, 2\rangle, \dots\}$, údaje v T^2 sú $\mathrm{D}_{T^2} = \{\langle 1, \texttt{Za siedmimi horami a siedmimi dolami}\rangle, \langle 2, \texttt{Kalifát Bagdad}\rangle, \dots\}$.

Pre metadáta potom podľa definície platí:

- $\mathrm{m}_J = \langle s_1^1, s_2^1, s_3^1\rangle \,||\, \langle s_1^2, s_2^2\rangle = \langle s_1^1, s_2^1, s_3^1, s_1^2, s_2^2\rangle$,
- $\mathrm{m}_L = \langle s_1^1, s_2^1, s_3^1\rangle \,||\, \langle t_1^2, t_2^2\rangle = \langle s_1^1, s_2^1, s_3^1, t_1^2, t_2^2\rangle$,
- $\mathrm{m}_R = \langle t_1^1, t_2^1, t_3^1\rangle \,||\, \langle s_1^2, s_2^2\rangle = \langle t_1^1, t_2^1, t_3^1, s_1^2, s_2^2\rangle$,
- $\mathrm{m}_F = \langle t_1^1, t_2^1, t_3^1\rangle \,||\, \langle t_1^2, t_2^2\rangle = \langle t_1^1, t_2^1, t_3^1, t_1^2, t_2^2\rangle$.

A dáta vyzerajú takto:

- Keďže $k = 1$, $j_1^1 = 3$ a $j_1^2 = 1$, podľa definície máme

$$\begin{aligned}\mathrm{D}_J &= \{r^1 \,||\, r^2 : (r^1 \in \mathrm{D}_{T^1}) \wedge (r^2 \in \mathrm{D}_{T^2}) \wedge (r_3^1 = r_1^2)\}\\ &= \{\langle \texttt{Ján}, \texttt{Hraško}, 1\rangle \,||\, \langle 1, \texttt{Za siedmimi horami a siedmimi dolami}\rangle,\\ &\quad\ \langle \texttt{Ružena}, \texttt{Šípová}, 1\rangle \,||\, \langle 1, \texttt{Za siedmimi horami a siedmimi dolami}\rangle,\\ &\quad\ \langle \texttt{Aladár}, \texttt{Baba}, 2\rangle \,||\, \langle 2, \texttt{Kalifát Bagdad}\rangle, \dots\}\\ &= \{\langle \texttt{Ján}, \texttt{Hraško}, 1, 1, \texttt{Za siedmimi horami a siedmimi dolami}\rangle,\\ &\quad\ \langle \texttt{Ružena}, \texttt{Šípová}, 1, 1, \texttt{Za siedmimi horami a siedmimi dolami}\rangle,\\ &\quad\ \langle \texttt{Aladár}, \texttt{Baba}, 2, 2, \texttt{Kalifát Bagdad}\rangle, \dots\}.\end{aligned}$$

- Použijeme tie r^1 z D_{T^1}, pre ktoré neexistuje žiadne r^2 z D_{T^2} také, že $r^1 \,||\, r^2 \in \mathrm{D}_J$, teda také, že platí $r^1.\texttt{id\_bydlisko} = r^2.\texttt{id}$. Sú to práve riadky $\langle \texttt{Jana}, \texttt{Botková}, \texttt{NULL}\rangle$, $\langle \texttt{Dana}, \texttt{Botková}, \texttt{NULL}\rangle$, $\langle \texttt{Aladár}, \texttt{Miazga}, \texttt{NULL}\rangle$ a $\langle \texttt{Jozef}, \texttt{Námorník}, \texttt{NULL}\rangle$. Keď na tieto riadky konkatenujeme sprava dvojicu $\langle \texttt{NULL}, \texttt{NULL}\rangle$, dostaneme pätice, ktoré treba dodať k práve uvedenej množine D_J, aby sme dostali výsledné D_L.
- Teraz treba tie r^2 z D_{T^2}, pre ktoré neexistuje žiadne r^1 z D_{T^1} také, že $r^1 \,||\, r^2 \in \mathrm{D}_J$, teda také, že $r^1.\texttt{id\_bydlisko} = r^2.\texttt{id}$. Taký riadok je jediný, a to $\langle 6, \texttt{Haliganda}\rangle$, a keď ho konkatenujeme dozadu k $\langle \texttt{NULL}, \texttt{NULL}, \texttt{NULL}\rangle$, dostaneme päticu, ktorá tvorí rozdiel hľadanej množiny D_R a D_J.
- D_F bude okrem riadkov D_J obsahovať oba spomínané prídavky.

Grafická podoba našich štyroch spojení je teda takáto:

- Vnútorné spojenie – $\iota(T^1, T^2, P) = \iota(T^1, T^2, \{\langle 3, 1\rangle\})$:

meno	priezvisko	id_bydlisko	id	názov
Reťazec	Reťazec	CeléČíslo	CeléČíslo	Reťazec
0	0	1	0	0
Ján	Hraško	1	1	Za siedmimi horami...
Ružena	Šípová	1	1	Za siedmimi horami...
Ján	Polienko	1	1	Za siedmimi horami...
Juraj	Truľo	1	1	Za siedmimi horami...
Ján	Hlúpy	1	1	Za siedmimi horami...
Aladár	Baba	2	2	Kalifát Bagdad
Ferdinand	Mravec	3	3	Mravenisko
Mikuláš	Myšiak	4	4	Hollywood
Donald	Káčer	4	4	Hollywood
Peter	Pan	5	5	Neverland

- Ľavé vonkajšie spojenie – $\iota^{\mathrm{L}}(T^1, T^2, P) = \iota^{\mathrm{L}}(T^1, T^2, \{\langle 3, 1\rangle\})$:

meno	priezvisko	id_bydlisko	id	názov
Reťazec	Reťazec	CeléČíslo	CeléČíslo	Reťazec
0	0	1	1	1
Ján	Hraško	1	1	Za siedmimi horami...
Ružena	Šípová	1	1	Za siedmimi horami...
Ján	Polienko	1	1	Za siedmimi horami...
Juraj	Truľo	1	1	Za siedmimi horami...
Ján	Hlúpy	1	1	Za siedmimi horami...
Aladár	Baba	2	2	Kalifát Bagdad
Ferdinand	Mravec	3	3	Mravenisko
Mikuláš	Myšiak	4	4	Hollywood
Donald	Káčer	4	4	Hollywood
Peter	Pan	5	5	Neverland
Jana	Botková	NULL	NULL	NULL
Dana	Botková	NULL	NULL	NULL
Aladár	Miazga	NULL	NULL	NULL
Jozef	Námorník	NULL	NULL	NULL

- Pravé vonkajšie spojenie – $\iota^{\mathrm{R}}(T^1, T^2, P) = \iota^{\mathrm{R}}(T^1, T^2, \{\langle 3, 1\rangle\})$:

meno	priezvisko	id_bydlisko	id	názov
Reťazec	Reťazec	CeléČíslo	CeléČíslo	Reťazec
1	1	1	0	0
Ján	Hraško	1	1	Za siedmimi horami...
Ružena	Šípová	1	1	Za siedmimi horami...
Ján	Polienko	1	1	Za siedmimi horami...
Juraj	Truľo	1	1	Za siedmimi horami...
Ján	Hlúpy	1	1	Za siedmimi horami...
Aladár	Baba	2	2	Kalifát Bagdad
Ferdinand	Mravec	3	3	Mravenisko
Mikuláš	Myšiak	4	4	Hollywood
Donald	Káčer	4	4	Hollywood
Peter	Pan	5	5	Neverland
NULL	NULL	NULL	6	Haliganda

- Úplné vonkajšie spojenie – $\iota^{\mathrm{F}}(T^1, T^2, P) = \iota^{\mathrm{F}}(T^1, T^2, \{\langle 3, 1\rangle\})$:

meno	priezvisko	id_bydlisko	id	názov
Reťazec	Reťazec	CeléČíslo	CeléČíslo	Reťazec
1	1	1	1	1
Ján	Hraško	1	1	Za siedmimi horami...
Ružena	Šípová	1	1	Za siedmimi horami...
Ján	Polienko	1	1	Za siedmimi horami...
Juraj	Truľo	1	1	Za siedmimi horami...
Ján	Hlúpy	1	1	Za siedmimi horami...
Aladár	Baba	2	2	Kalifát Bagdad
Ferdinand	Mravec	3	3	Mravenisko
Mikuláš	Myšiak	4	4	Hollywood
Donald	Káčer	4	4	Hollywood
Peter	Pan	5	5	Neverland
NULL	NULL	NULL	6	Haliganda
Jana	Botková	NULL	NULL	NULL
Dana	Botková	NULL	NULL	NULL
Aladár	Miazga	NULL	NULL	NULL
Jozef	Námorník	NULL	NULL	NULL

Všimnime si, že ak je množina P prázdna, tabuľky nemajú byť cez čo prepojené, a výsledkom (vnútorného) spojenia je ich **karteziánsky súčin**, ktorého výsledkom sú všetky konkatenácie riadkov z T^1 a riadkov z T^2. Skrátene túto operáciu môžeme zapísať

$$\kappa(T^1, T^2) = \iota(T^1, T^2, \emptyset).$$

4.3.4 Zjednotenie, prienik, rozdiel

Spomeňme ešte množinové operácie, budeme teda typovanými reláciami simulovať dopyty `SELECT * FROM` T^1 `UNION`/`INTERSECT`/`EXCEPT SELECT * FROM` T^2. Opäť si uvedomme, že v prípade typovaných relácií musíme rezignovať na `ALL`.

Definícia

- Nech T^1 a T^2 sú typované relácie, pričom $\mathrm{m}_{T^1} = \langle s_1^1, \ldots, s_n^1\rangle$, $\mathrm{m}_{T^2} = \langle s_1^2, \ldots, s_n^2\rangle$ a pre každé $i \in \{1, \ldots, n\}$ platí $s_i^1.\mathrm{názov} = s_i^2.\mathrm{názov}$ a $s_i^1.\mathrm{typ} = s_i^2.\mathrm{typ}$ (t. j. navzájom si zodpovedajúce stĺpce sa môžu líšiť iba v nullabilite). Potom definujme tieto operácie:
  - **Zjednotenie** $T^1 \cup T^2$ bude typovaná relácia U, ktorej dáta sú

    $$\mathrm{D}_U = \mathrm{D}_{T^1} \cup \mathrm{D}_{T^2}$$

    a metadáta

    $$\mathrm{m}_U = \langle t_1^{\max}, \ldots, t_n^{\max}\rangle$$

    také, že pre každé $i \in \{1, \ldots, n\}$ platí:
    - $t_i^{\max}.\mathrm{názov} = s_i^1.\mathrm{názov} = s_i^2.\mathrm{názov}$,
    - $t_i^{\max}.\mathrm{typ} = s_i^1.\mathrm{typ} = s_i^2.\mathrm{typ}$,
    - $t_i^{\max}.\mathrm{nullabilita} = \max\{s_i^1.\mathrm{nullabilita}, s_i^2.\mathrm{nullabilita}\}$ (t. j. prázdne hodnoty sú vo výsledku prípustné vtedy, ak sú prípustné čo len v jednom operande).

- **Prienik** $T^1 \cap T^2$ bude typovaná relácia I, ktorej dáta sú

  $$\mathrm{D}_I = \mathrm{D}_{T^1} \cap \mathrm{D}_{T^2}$$

  a metadáta

  $$\mathrm{m}_I = \langle t_1^{\min}, \ldots, t_n^{\min} \rangle$$

  také, že pre každé $i \in \{1, \ldots, n\}$ platí:
  - $t_i^{\min}.\text{názov} = s_i^1.\text{názov} = s_i^2.\text{názov}$,
  - $t_i^{\min}.\text{typ} = s_i^1.\text{typ} = s_i^2.\text{typ}$,
  - $t_i^{\min}.\text{nullabilita} = \min\{s_i^1.\text{nullabilita}, s_i^2.\text{nullabilita}\}$ (t. j. prázdne hodnoty sú vo výsledku prípustné len vtedy, ak sú prípustné v oboch operandoch).
- **Rozdiel** $T^1 \setminus T^2$ bude typovaná relácia E, ktorej dáta sú $\mathrm{D}_E = \mathrm{D}_{T^1} \setminus \mathrm{D}_{T^2}$ a metadáta sú $\mathrm{m}_E = \mathrm{m}_{T^1}$.

Opäť asi bude užitočné osvetliť tieto označenia na príklade. Nech T^1 je typovaná relácia obsahujúca mená a bydliská prvákov:

meno	bydlisko
Reťazec	Reťazec
0	0
Ján	Za siedmimi horami a siedmimi dolami
Ružena	Za siedmimi horami a siedmimi dolami
Juraj	Za siedmimi horami a siedmimi dolami
Peter	Neverland

a T^2 to isté, ale pre druhákov:

meno	bydlisko
Reťazec	Reťazec
0	1
Ján	Za siedmimi horami a siedmimi dolami
Aladár	Kalifát Bagdad
Jozef	NULL

Potom platí:

- Počty stĺpcov sú v oboch typovaných reláciách zhodne $n = 2$.
- Stĺpce relácie T^1 sú $s_1^1 = \langle \texttt{meno}, \text{Reťazec}, 0 \rangle$ a $s_2^1 = \langle \texttt{bydlisko}, \text{Reťazec}, 0 \rangle$, stĺpce T^2 sú $s_1^2 = \langle \texttt{meno}, \text{Reťazec}, 0 \rangle$ a $s_2^2 = \langle \texttt{bydlisko}, \text{Reťazec}, 1 \rangle$ (platí teda $s_1^1 = s_1^2$ a dvojica s_2^1 a s_2^2 sa navzájom líši len nullabilitou). Potom zrejme $\mathrm{m}_{T^1} = \langle s_1^1, s_2^1 \rangle$ a $\mathrm{m}_{T^2} = \langle s_1^2, s_2^2 \rangle$.
- Pre stĺpce zjednotenia U a prieniku I potom platí:
  - $t_1^{\max}.\text{názov} = t_1^{\min}.\text{názov} = s_1^1.\text{názov} = s_1^2.\text{názov} = \texttt{meno}$,
  - $t_2^{\max}.\text{názov} = t_2^{\min}.\text{názov} = s_2^1.\text{názov} = s_2^2.\text{názov} = \texttt{bydlisko}$,
  - $t_1^{\max}.\text{typ} = t_1^{\min}.\text{typ} = s_1^1.\text{typ} = s_1^2.\text{typ} = \text{Reťazec}$,
  - $t_2^{\max}.\text{typ} = t_2^{\min}.\text{typ} = s_2^1.\text{typ} = s_2^2.\text{typ} = \text{Reťazec}$,
  - $t_1^{\max}.\text{nullabilita} = \max\{s_1^1.\text{nullabilita}, s_1^2.\text{nullabilita}\} = \max\{0, 0\} = 0$,
  - $t_2^{\max}.\text{nullabilita} = \max\{s_2^1.\text{nullabilita}, s_2^2.\text{nullabilita}\} = \max\{0, 1\} = 1$,

- $t_1^{\min}.\text{nullabilita} = \min\{s_1^1.\text{nullabilita}, s_1^2.\text{nullabilita}\} = \min\{0,0\} = 0$,
- $t_2^{\min}.\text{nullabilita} = \min\{s_2^1.\text{nullabilita}, s_2^2.\text{nullabilita}\} = \min\{0,1\} = 0$.

Z toho $\mathrm{m}_U = \langle t_1^{\max}, t_2^{\max}\rangle = \langle\langle \texttt{meno}, \text{Reťazec}, 0\rangle, \langle \texttt{bydlisko}, \text{Reťazec}, 1\rangle\rangle$ a $\mathrm{m}_I = \langle t_1^{\min}, t_2^{\min}\rangle = \langle\langle \texttt{meno}, \text{Reťazec}, 0\rangle, \langle \texttt{bydlisko}, \text{Reťazec}, 0\rangle\rangle$.

- Metadáta rozdielu E sú identické s metadátami relácie T^1, t. j. $\mathrm{m}_E = \langle s_1^1, s_2^1\rangle = \langle\langle \texttt{meno}, \text{Reťazec}, 0\rangle, \langle \texttt{bydlisko}, \text{Reťazec}, 0\rangle\rangle$.
- Pokiaľ ide o dáta všetkých troch nových relácií, tie vzniknú praobyčajnými množinovými operáciami na množinách D_{T^1} a D_{T^2}.

Prehľadnejšie bude zobraziť naše výsledky v už klasickej podobe tabuliek. Najprv zjednotenie (všimnime si, že záznam $\langle \texttt{Ján}, \texttt{Za siedmimi horami a siedmimi dolami}\rangle$ sa vo výsledku ocitne iba raz, veď ten je množina):

meno	bydlisko
Reťazec	Reťazec
0	1
Ján	Za siedmimi horami a siedmimi dolami
Ružena	Za siedmimi horami a siedmimi dolami
Juraj	Za siedmimi horami a siedmimi dolami
Peter	Neverland
Aladár	Kalifát Bagdad
Jozef	NULL

Teraz prienik:

meno	bydlisko
Reťazec	Reťazec
0	0
Ján	Za siedmimi horami a siedmimi dolami

A napokon rozdiel:

meno	bydlisko
Reťazec	Reťazec
0	0
Ružena	Za siedmimi horami a siedmimi dolami
Juraj	Za siedmimi horami a siedmimi dolami
Peter	Neverland

4.3.5 Skladanie operácií

Doposiaľ nás pri každom dopyte zaujímala len odpoveď naň, proces jej získania pre nás ostával zahalený (nepochybne aj obchodným) tajomstvom (a najskôr sme si ho ani neuvedomovali). Skúsme teraz túto čiernu skrinku pootvoriť. Ako pri ľubovoľnom inom výpočte, aj tu zrejme pôjde o sled akýchsi krokov, ktorý nazveme (ako obvykle) **vytvárajúca postupnosť**. Každý jej krok produkuje akýsi medzivýsledok, pričom posledný z nich bude zároveň požadovanou odpoveďou. Prirodzenou zásadou tu je, že pred vytváraním každého zložitejšieho medzivýsledku už treba mať v postupnosti pripravené všetky jeho zložky. Vytvárajúcu postupnosť tak možno vo všeobecnosti považovať za akýsi „recept“, keď zo základných „surovín“ použitím istých „úkonov“ v určitom

poradí dostaneme požadovaný produkt. V našom prípade sú takými surovinami typované relácie zodpovedajúce tabuľkám našej databázy a úkonmi sú algebraické operácie spomínané v statiach 4.3.1 až 4.3.4. Keďže výsledkom každého kroku každej vytvárajúcej postupnosti je akási typovaná relácia, proces jej vytvárania nazveme **relačná algebra**.

Ukážme si hľadanie vytvárajúcej postupnosti na príklade. Vráťme sa opäť k našej univerzite, a to dopytom, ktorý požaduje v jednom zozname osoby s priezviskom začínajúcim na písmeno M, pričom pri študentoch chceme ich ročník a študijnú skupinu a pri učiteľoch titul. Tento dopyt (bez podmienky na začiatočné písmeno) sme už zdôvodňovali, preto ho viac-menej len zopakujeme (použijeme však pri tom DISTINCT a UNION):

```
SELECT DISTINCT t.skratka || ' ' || o.priezvisko AS osoba_na_M
FROM
  osoba AS o
    JOIN učiteľ AS u ON o.id = u.id
      JOIN titul AS t ON u.id_titul = t.id
WHERE SUBSTR(o.priezvisko,1,1) = 'M'

UNION

SELECT DISTINCT o.priezvisko || ' ' || '(' || RTRIM(CHAR(š.ročník)) || RTRIM(s.kód) || ')' AS osoba_na_M
FROM
  osoba AS o
    JOIN študent AS š ON o.id = š.id
      JOIN študijná_skupina AS s ON š.id_skupina = s.id
WHERE SUBSTR(o.priezvisko,1,1) = 'M'
```

Jeho vytvárajúcou postupnosťou bude:

0 Začneme centrálnou tabuľkou potrebnou v oboch častiach zjednotenia. Je to jedna z našich surovín: $R_0 =$ osoba:

id	meno	priezvisko	pohlavie	dátum_narodenia
CeléČíslo	Reťazec	Reťazec	Reťazec	Dátum
0	0	0	0	0
1	Gejza	Miazga	muž	12.12.1955
2	Matej	Múdry	muž	11.6.1945
3	Vasilisa	Premúdra	žena	22.1.1973
4	Hedviga	Baba	žena	3.5.1784
5	d'Eduard	Vševed	muž	13.3.1900
101	Ján	Hraško	muž	11.7.1987
102	Ružena	Šípová	žena	1.2.1984
103	Aladár	Baba	muž	22.1.1980
104	Ferdinand	Mravec	muž	3.3.1984
105	Ján	Polienko	muž	14.4.1982
106	Juraj	Truľo	muž	16.7.1979
107	Jana	Botková	žena	21.9.1977
108	Dana	Botková	žena	21.9.1977
109	Ján	Hlúpy	muž	1.4.1972
110	Aladár	Miazga	muž	22.12.1987
111	Mikuláš	Myšiak	muž	6.6.1983
112	Donald	Káčer	muž	7.10.1982
113	Jozef	Námorník	muž	23.9.1981
114	Peter	Pan	muž	13.1.2001

1 V oboch častiach zjednotenia máme tú istú podmienku, aplikujme teda na R_0 selekciu (pričom zjednodušíme kvalifikovaný názov stĺpca na priezvisko): $R_1 = \sigma(R_0, \texttt{SUBSTR(priezvisko,1,1)='M'})$:

id	meno	priezvisko	pohlavie	dátum_narodenia
CeléČíslo	Reťazec	Reťazec	Reťazec	Dátum
0	0	0	0	0
1	Gejza	Miazga	muž	12.12.1955
2	Matej	Múdry	muž	11.6.1945
104	Ferdinand	Mravec	muž	3.3.1984
110	Aladár	Miazga	muž	22.12.1987
111	Mikuláš	Myšiak	muž	6.6.1983

2 Ako v prvej, tak v druhej časti dopytu pracujeme len s dvoma stĺpcami typovanej relácie osoba, a to s id a priezvisko. Aplikujme teda projekciu:
$R_2 = \pi(R_1, \langle 1, 3 \rangle)$:

id	priezvisko
CeléČíslo	Reťazec
0	0
1	Miazga
2	Múdry
104	Mravec
110	Miazga
111	Myšiak

3 Sústreďme sa teraz na prvú časť dopytu, ktorá hovorí o učiteľoch. Potrebujeme teda rovnomennú tabuľku, ktorá bude našou ďalšou surovinou:
$R_3 =$ učiteľ:

id	id_titul
CeléČíslo	CeléČíslo
0	0
1	1
2	1
3	2
4	2
5	3

4 Ak ju chceme napojiť na reláciu R_2, mali by sme formálne premenovať jej stĺpec id, aby výsledné spojenie bolo typovanou reláciou s rôznymi názvami stĺpcov (treba však povedať, že tento krok nie je nutný, robíme ho len kvôli prehľadnosti medzivýsledku):
$R_4 = \varrho(R_3, \{\langle 1, \text{u\_id} \rangle\})$:

u_id	id_titul
CeléČíslo	CeléČíslo
0	0
1	1
2	1
3	2
4	2
5	3

5 A môžeme napájať. Prepájacími stĺpcami budú id z R_2 (čo je jej 1. stĺpec) a u_id z R_4 (aj tu je to 1. stĺpec):
$R_5 = \iota(R_2, R_4, \{\langle 1, 1 \rangle\})$:

id	priezvisko	u_id	id_titul
CeléČíslo	Reťazec	CeléČíslo	CeléČíslo
0	0	0	0
1	Miazga	1	1
2	Múdry	2	1

6 Vzhľadom na nepotrebnosť stĺpca `u_id` (aj tak sa jeho hodnoty z definície R_5 rovnajú hodnotám `id`) vyberieme len ostatné tri:
$R_6 = \pi(R_5, \langle 1, 2, 4 \rangle)$:

id	priezvisko	id_titul
CeléČíslo	Reťazec	CeléČíslo
0	0	0
1	Miazga	1
2	Múdry	1

7 Ďalšou tabuľkou zaangažovanou v prvej časti je `titul`:
$R_7 =$ `titul`:

id	skratka	názov
CeléČíslo	Reťazec	Reťazec
0	0	0
1	.doc	doc(um)ent
2	Mgr.	managor
3	DEd	doctor of education

8 Z nej nám treba len prvé dva stĺpce, preto použijeme projekciu:
$R_8 = \pi(R_7, \langle 1, 2 \rangle)$:

id	skratka
CeléČíslo	Reťazec
0	0
1	.doc
2	Mgr.
3	DEd

9 Opäť premenujeme aspoň stĺpec `id` potrebný na prepájanie:
$R_9 = \varrho(R_8, \{\langle 1, \texttt{t\_id} \rangle\})$:

t_id	skratka
CeléČíslo	Reťazec
0	0
1	.doc
2	Mgr.
3	DEd

10 Opäť môžeme spájať. Teraz budú prepájacími stĺpcami `id_titul` z R_6 (jej 3. stĺpec) a `t_id` z R_9 (1. stĺpec):
$R_{10} = \iota(R_6, R_9, \{\langle 3, 1 \rangle\})$:

id	priezvisko	id_titul	t_id	skratka
CeléČíslo	Reťazec	CeléČíslo	CeléČíslo	Reťazec
0	0	0	0	0
1	Miazga	1	1	.doc
2	Múdry	1	1	.doc

11 Prvú časť uzavrieme transformáciou stĺpcov (opäť skrátime kvalifikované názvy použitých stĺpcov):
$R_{11} = \tau(R_{10}, \langle\langle$`skratka || ' ' || priezvisko`$,$`osoba_na_M`$\rangle\rangle)$:

osoba_na_M
Reťazec
0
.doc Miazga
.doc Múdry

12 Poďme na druhú časť:
$R_{12} =$ `študent`:

id	id_skupina	ročník	id_krajina	id_izba
CeléČíslo	CeléČíslo	CeléČíslo	CeléČíslo	CeléČíslo
0	0	0	1	1
101	1	1	1	NULL
102	1	1	1	1
103	2	2	2	4
104	2	3	3	3
105	3	5	1	3
106	3	1	1	3
107	3	4	NULL	1
108	3	4	NULL	1
109	2	2	1	2
110	2	3	NULL	4
111	1	5	4	5
112	1	5	4	5
113	1	2	NULL	NULL
114	2	1	5	NULL

13 Vyberieme relevantné stĺpce:
$R_{13} = \pi(R_{12}, \langle 1, 2, 3\rangle)$:

id	id_skupina	ročník
CeléČíslo	CeléČíslo	CeléČíslo
0	0	0
101	1	1
102	1	1
103	2	2
104	2	3
105	3	5
106	3	1
107	3	4
108	3	4
109	2	2
110	2	3
111	1	5
112	1	5
113	1	2
114	2	1

14 Pred napojením radšej premenujeme `id`:
$R_{14} = \varrho(R_{13}, \{\langle 1,$ `š_id`$\rangle\})$:

š_id	id_skupina	ročník
CeléČíslo	CeléČíslo	CeléČíslo
0	0	0
101	1	1
102	1	1
103	2	2
104	2	3
105	3	5
106	3	1
107	3	4
108	3	4
109	2	2
110	2	3
111	1	5
112	1	5
113	1	2
114	2	1

15 Napojíme na torzo tabuľky osoba, a to cez stĺpce id (1. stĺpec R_2) a š_id (1. stĺpec R_{14}):
$R_{15} = \iota(R_2, R_{14}, \{\langle 1, 1\rangle\})$:

id	priezvisko	š_id	id_skupina	ročník
CeléČíslo	Reťazec	CeléČíslo	CeléČíslo	CeléČíslo
0	0	0	0	0
104	Mravec	104	2	3
110	Miazga	110	2	3
111	Myšiak	111	1	5

16 Zbavme sa zbytočného stĺpca š_id:
$R_{16} = \pi(R_{15}, \langle 1, 2, 4, 5\rangle)$:

id	priezvisko	id_skupina	ročník
CeléČíslo	Reťazec	CeléČíslo	CeléČíslo
0	0	0	0
104	Mravec	2	3
110	Miazga	2	3
111	Myšiak	1	5

17 Ešte posledná tabuľka:
$R_{17} =$ študijná_skupina:

id	kód	názov
CeléČíslo	Reťazec	Reťazec
0	0	0
1	M	metamatematika
2	I	informatematika
3	MI	metamatematika-informatematika

18 Zasa vyberieme len to, čo treba:
$R_{18} = \pi(R_{17}, \langle 1, 2\rangle)$:

id	kód
CeléČíslo	Reťazec
0	0
1	M
2	I
3	MI

19 Premenujeme:
$R_{19} = \varrho(R_{18}, \{\langle 1, \texttt{s\_id}\rangle\})$:

s_id	kód
CeléČíslo	Reťazec
0	0
1	M
2	I
3	MI

20 A naviažeme na doterajší medzivýsledok, tentoraz cez stĺpce `id_skupina` (3. stĺpec R_{16}) a `s_id` (1. stĺpec R_{19}):
$R_{20} = \iota(R_{16}, R_{19}, \{\langle 3, 1\rangle\})$:

id	priezvisko	id_skupina	ročník	s_id	kód
CeléČíslo	Reťazec	CeléČíslo	CeléČíslo	CeléČíslo	Reťazec
0	0	0	0	0	0
104	Mravec	2	3	2	I
110	Miazga	2	3	2	I
111	Myšiak	1	5	1	M

21 Aj druhú časť ukončíme transformáciou stĺpcov (oproti SQL so skrátenými názvami):
$R_{21} = \tau(R_{20}, \langle\langle \texttt{priezvisko||' ('||RTRIM(CHAR(ročník))||RTRIM(kód)||')'}, \texttt{osoba\_na\_M}\rangle\rangle)$:

osoba_na_M
Reťazec
0
Mravec (3I)
Miazga (3I)
Myšiak (5M)

22 A fanfáry k veľkému finále, spočívajúcom v zjednotení oboch častí:
$R_{22} = R_{11} \cup R_{21}$:

osoba_na_M
Reťazec
0
.doc Miazga
.doc Múdry
Mravec (3I)
Miazga (3I)
Myšiak (5M)

To, že k dopytu vieme nájsť jeho vytvárajúcu postupnosť, však ešte neznamená, že databázový stroj hľadá odpoveď presne takýmto spôsobom. Jednak môže byť vytvárajúcich postupností prislúchajúcich tomu istému dopytu viac, jednak môžu byť elementárne operácie databázového stroja komplikovanejšie. Nezabudnime ani

na to, že sme sa vyhli napríklad vnoreným dopytom, ba agregačné funkcie (ktoré presahujú logiku prvého rádu) sme dokonca ani nespomenuli. Do hry tiež vstupujú rôzne optimalizačné algoritmy, využívajúce napríklad indexy alebo znalosť o celkovom počte riadkov tej-ktorej tabuľky. Skutočnosť je teda omnoho komplikovanejšia, celkovo však možno povedať, že **hľadanie odpovede na dopyt sleduje nejakú vhodnú vytvárajúcu postupnosť v istej (i keď pravdepodobne nie práve našej) relačnej algebre**.

4.3.Ú Úlohy

1 Zistite, akú mohutnosť môže (a akú nemôže) mať množina

$$\{\iota(T^1, T^2, P), \iota^{\mathrm{L}}(T^1, T^2, P), \iota^{\mathrm{R}}(T^1, T^2, P), \iota^{\mathrm{F}}(T^1, T^2, P)\},$$

a svoje tvrdenie dokážte.

4.4 Funkčné závislosti a normálne formy

4.4.1 Funkčné závislosti

Pripomeňme, že po vybudovaní databázového modelu v podkapitole 2.4 bolo treba vykonať istú revíziu. Na to, že náš model nie je úplne v poriadku, nás upozornila funkčná závislosť medzi jednotlivými stĺpcami problematickej tabuľky. V tejto stati sa pokúsime tento pojem formalizovať.

Z technických dôvodov sa tu ukazuje výhodným použiť namiesto typovanej relácie abstraktnú tabuľku. Metadáta tabuľky T teda nebudú reprezentované ticou m_T, ale množinou M_T stĺpcov.

Najprv jedna pomocná všeobecná definícia:

Definícia

- Nech f^1 a f^2 sú funkcie a nech X je podmnožina ako definičného oboru f^1, tak definičného oboru f^2. Potom hovoríme, že f^1 a f^2 **sa zhodujú** na X, a píšeme $\mathrm{zhoda}(f^1, f^2, X)$, ak sa v každom prvku X hodnoty funkcií f^1 a f^2 rovnajú. Formálne to možno vyjadriť vzťahom

$$\mathrm{zhoda}(f^1, f^2, X), \quad \text{akk} \quad (\forall x \in X)\ f^1(x) = f^2(x).$$

My budeme túto definíciu používať pre záznamy abstraktných tabuliek (nezaškodí pripomenúť, že sú to funkcie na stĺpcoch). Napríklad ak máme abstraktnú tabuľku:

meno	priezvisko	pohlavie
Reťazec	Reťazec	Reťazec
0	0	0
Ružena	Šípová	žena
Jana	Botková	žena
Dana	Botková	žena

a stĺpec s názvom x označíme s_x (a tak to budeme robiť aj naďalej), tak záznam r^1 zodpovedajúci Ružene a záznam r^2 zodpovedajúci Jane sa zhodujú na množine $\{\mathrm{s}_{\texttt{pohlavie}}\}$, čiže platí $\mathrm{zhoda}(r^1, r^2, \{\mathrm{s}_{\texttt{pohlavie}}\})$. Podobne sa na tejto množine zhoduje r^2 so záznamom r^3 o Dane, takže platí aj $\mathrm{zhoda}(r^2, r^3, \{\mathrm{s}_{\texttt{pohlavie}}\})$. Tieto dva záznamy sa navyše zhodujú aj v stĺpci `priezvisko`, teda platí aj $\mathrm{zhoda}(r^2, r^3, \{\mathrm{s}_{\texttt{priezvisko}}, \mathrm{s}_{\texttt{pohlavie}}\})$. Avšak $\mathrm{zhoda}(r^1, r^2, \{\mathrm{s}_{\texttt{priezvisko}}, \mathrm{s}_{\texttt{pohlavie}}\})$ neplatí, keďže $r^1(\mathrm{s}_{\texttt{priezvisko}}) \neq r^2(\mathrm{s}_{\texttt{priezvisko}})$. Dodajme ešte, že všetky záznamy (a to nielen v tejto tabuľke) sa zrejme zhodujú na prázdnej množine.

Teraz už môžeme definovať funkčnú závislosť:

Definícia

- Nech T je abstraktná tabuľka, nech X a Y sú podmnožiny množiny jej stĺpcov. Potom hovoríme, že množina X **funkčne určuje** množinu Y alebo (ekvivalentne) že Y **je funkčne závislá** na X, a píšeme $X \to_T Y$, ak všetky dvojice záznamov r^1 a r^2 zhodujúce sa na množine X sa zhodujú aj na množine Y. Formálne teda

$$X \to_T Y, \quad \text{akk} \quad (\forall r^1, r^2 \in \mathrm{D}_T)\ \mathrm{zhoda}(r^1, r^2, X) \to \mathrm{zhoda}(r^1, r^2, Y).$$

  Dvojicu $\langle X, Y \rangle$ potom nazývame **funkčná závislosť**.
- Ak je množina Y jednoprvková s prvkom s, hovoríme o **elementárnej** funkčnej závislosti a okrem označenia $X \to_T \{s\}$ používame aj zjednodušené $X \to_T s$.
- Ak $Y \subseteq X$, vzťah $X \to_T Y$ budeme nazývať **triviálna** funkčná závislosť.

V tabuľke z predchádzajúceho príkladu (označme ju T) vidíme hneď niekoľko funkčných závislostí. Vzhľadom na to, že všetky hodnoty v stĺpci $\mathtt{s_{meno}}$ sú rôzne, sú od neho funkčne závislé všetky množiny stĺpcov, platí teda napríklad $\{\mathtt{s_{meno}}\} \to_T \{\mathtt{s_{priezvisko}}, \mathtt{s_{pohlavie}}\}$, ale aj elementárne závislosti $\{\mathtt{s_{meno}}\} \to_T \mathtt{s_{priezvisko}}$ či $\{\mathtt{s_{meno}}\} \to_T \mathtt{s_{pohlavie}}$ (definícia nám vzhľadom na jednoprvkovosť množiny na pravej strane povoľuje vynechať zátvorky). Najilustratívnejšou funkčnou závislosťou v tejto tabuľke je však $\{\mathtt{s_{priezvisko}}\} \to_T \mathtt{s_{pohlavie}}$, keďže nie všetky priezviská sú rôzne. Pokiaľ ide o triviálne funkčné závislosti (ako ukážky poslúžia trebárs $\{\mathtt{s_{meno}}, \mathtt{s_{priezvisko}}, \mathtt{s_{pohlavie}}\} \to_T \mathtt{s_{priezvisko}}$ alebo $\{\mathtt{s_{pohlavie}}\} \to_T \mathtt{s_{pohlavie}}$), uvedomme si, že sú splnené automaticky. Naproti tomu neplatí ani závislosť $\{\mathtt{s_{pohlavie}}\} \to_T \mathtt{s_{priezvisko}}$ (lebo platí $\mathrm{zhoda}(r^1, r^2, \{\mathtt{s_{pohlavie}}\})$, ale nie $\mathrm{zhoda}(r^1, r^2, \{\mathtt{s_{priezvisko}}\})$), ani závislosť $\{\mathtt{s_{priezvisko}}\} \to_T \mathtt{s_{meno}}$ (keďže $\mathrm{zhoda}(r^2, r^3, \{\mathtt{s_{priezvisko}}\})$ platí, ale $\mathrm{zhoda}(r^2, r^3, \{\mathtt{s_{meno}}\})$ nie).

Pre funkčné závislosti môžeme vysloviť niekoľko jednoduchých tvrdení nazývaných **Armstrongove pravidlá**:

Lema

- Nech T je abstraktná tabuľka a nech $X, Y, Z, Y_1, Y_2 \subseteq \mathrm{M}_T$. Potom platí:
  - 1 Ak $Y \subseteq X$, tak $X \to_T Y$ (t. j. každá triviálna funkčná závislosť je splnená).
  - 2a Ak $X \to_T Y$ a $Y \to_T Z$, tak $X \to_T Z$ (**tranzitivita**).
  - 2b1 Ak $X \to_T Y_1$ a $X \to_T Y_2$, tak $X \to_T (Y_1 \cup Y_2)$ (**kompozícia**).
  - 2b2 Ak $X \to_T (Y_1 \cup Y_2)$, tak $X \to_T Y_1$ a $X \to_T Y_2$ (**dekompozícia**).

Za zmienku určite stojí postreh, že pravidlá 2b1 a 2b2 nám umožňujú každú neelementárnu závislosť ekvivalentne rozložiť na niekoľko zároveň platiacich elementárnych. Formálne teda dostávame:

Dôsledok

- Nech T je abstraktná tabuľka a nech $X, Y \subseteq \mathrm{M}_T$. Potom

$$X \to_T Y, \quad \text{akk} \quad (\forall s \in Y)\ X \to_T s.$$

Na pravej strane pritom môžeme ignorovať (podľa 1. Armstrongovho pravidla automaticky splnené) vzťahy $X \to_T s$ pre všetky $s \in X$. Z predchádzajúceho vzťahu potom dostávame túto ekvivalenciu:

Dôsledok

- Nech T je abstraktná tabuľka a nech $X, Y \subseteq \mathrm{M}_T$. Potom

$$X \to_T Y, \quad \text{akk} \quad X \to_T (Y \setminus X).$$

Stačí sa teda obmedziť na závislosti, ktorých obe strany sú disjunktné.

4.4.2 Schéma abstraktných tabuliek

Doteraz sme hovorili o funkčných závislostiach v danej abstraktnej tabuľke, no smer našich úvah môžeme aj obrátiť, a to tak, že najprv vyslovíme ako podmienku isté funkčné závislosti, a až potom hľadáme relácie (s dopredu danými metadátami), ktoré ich spĺňajú. Všimnime si, že takýto prístup korešponduje s myšlienkou vytvárania tabuľky v SQL, keď najprv definujeme nielen stĺpce tabuľky, ale aj jej kľúče (ako špeciálny prípad funkčných závislostí), a až potom sa zaoberáme samotnými dátami tabuľky. Napriek tomu, že potom už takúto tabuľku chápeme ako stále to isté indivíduum (dokonca jej dávame meno), pri každej zmene dát ide z formálneho hľadiska vždy o novú a novú abstraktnú tabuľku. Jediné, čo je na nej v čase nemenné, sú jej metadáta (a deklarované obmedzenia). Túto myšlienku vyjadríme v nasledujúcej definícii:

Definícia

- Nech S je množina stĺpcov s rôznymi menami a F je množina dvojíc jej podmnožín. Pod označením $\mathrm{schéma}(S, F)$ budeme rozumieť množinu všetkých abstraktných tabuliek T spĺňajúcich podmienky:

1 $\mathrm{M}_T = S$.

2 Pre každú dvojicu $\langle X, Y\rangle \in F$ platí $X \to_T Y$.

Túto množinu budeme nazývať **schéma** abstraktných tabuliek daná stĺpcami z S a závislosťami z F.

- Ak vlastnosť 2 platí pre každú abstraktnú tabuľku T z množiny $\mathrm{schéma}(S, F)$, budeme to označovať $X \to_{S,F} Y$ (resp. skrátene $X \to Y$), t. j. formálne

$$X \to_{S,F} Y, \quad \text{akk} \quad (\forall T \in \mathrm{schéma}(S, F))\ X \to_T Y.$$

Vzhľadom na predchádzajúce tvrdenie o dekompozícii závislej množiny pritom stačí predpokladať, že vo všetkých dvojiciach z F je druhá zložka jednoprvková (a teda všetky závislosti z F sú elementárne).

Ak je $S = \{\langle \texttt{meno}, \mathrm{Reťazec}, 0\rangle, \langle \texttt{priezvisko}, \mathrm{Reťazec}, 0\rangle, \langle \texttt{pohlavie}, \mathrm{Reťazec}, 0\rangle\}$ množina stĺpcov a $F = \{\langle\{\mathrm{s}_{\texttt{meno}}\}, \{\mathrm{s}_{\texttt{priezvisko}}\}\rangle, \langle\{\mathrm{s}_{\texttt{priezvisko}}\}, \{\mathrm{s}_{\texttt{pohlavie}}\}\rangle\}$ množina dvojíc jej podmnožín (teda deklarovaných funkčných závislostí), tak do schémy $\mathrm{schéma}(S, F)$ patrí abstraktná tabuľka z predchádzajúcej state, ale napríklad aj abstraktná tabuľka:

meno	priezvisko	pohlavie
Reťazec	Reťazec	Reťazec
0	0	0
Gejza	Miazga	muž
Matej	Múdry	muž
Vasilisa	Premúdra	žena
Hedviga	Baba	žena
d'Eduard	Vševed	muž

Naproti tomu abstraktná tabuľka:

meno	priezvisko	pohlavie
Reťazec	Reťazec	Reťazec
0	0	0
Ján	Hraško	muž
Ružena	Šípová	žena
Aladár	Baba	muž
Ferdinand	Mravec	muž
Ján	Polienko	muž
Juraj	Truľo	muž
Jana	Botková	žena
Dana	Botková	žena
Ján	Hlúpy	muž
Aladár	Miazga	muž
Mikuláš	Myšiak	muž
Donald	Káčer	muž
Jozef	Námorník	muž
Peter	Pan	muž

do tejto schémy nepatrí, lebo nespĺňa deklarovanú (ale, priznajme si, sémanticky dosť pochybnú) závislosť $\langle\{\mathrm{s}_{\texttt{meno}}\}, \{\mathrm{s}_{\texttt{priezvisko}}\}\rangle$. Ďalším príkladom abstraktnej tabuľky mimo našej schémy je trebárs:

meno	priezvisko	pohlavie
Reťazec	Reťazec	Reťazec
0	0	0
Hedviga	Baba	žena
Aladár	Baba	muž

keďže pre ňu neplatí požadovaná závislosť $\langle\{s_{\texttt{priezvisko}}\},\{s_{\texttt{pohlavie}}\}\rangle$ z množiny F. Kontrapríkladom nespĺňajúcim 1. bod definície je napríklad abstraktná tabuľka:

meno	priezvisko	pohlavie	dátum_narodenia
Reťazec	Reťazec	Reťazec	Dátum
0	0	0	0
Ružena	Šípová	žena	1.2.1984
Jana	Botková	žena	21.9.1977
Dana	Botková	žena	21.9.1977

aj keď v nej platia obe deklarované závislosti z F.

Podľa predchádzajúcej definície z $\langle X,Y\rangle \in F$ zrejme vyplýva $X \to_{S,F} Y$. Všimnime si to teraz, ako je to s opačným smerom. Keďže pre každú tabuľku zo schémy platia Armstrongove pravidlá, ľahko vidieť ich platnosť aj pre celú schému:

Lema

- Nech S je množina stĺpcov s rôznymi menami, F je množina dvojíc jej podmnožín a nech ďalej $X,Y,Z,Y_1,Y_2 \subseteq S$. Potom v množine $\text{schéma}(S,F)$ platí:
  - 1 Ak $Y \subseteq X$, tak $X \to_{S,F} Y$.
  - 2a Ak $X \to_{S,F} Y$ a $Y \to_{S,F} Z$, tak $X \to_{S,F} Z$.
  - 2b1 Ak $X \to_{S,F} Y_1$ a $X \to_{S,F} Y_2$, tak $X \to_{S,F} (Y_1 \cup Y_2)$.
  - 2b2 Ak $X \to_{S,F} (Y_1 \cup Y_2)$, tak $X \to_{S,F} Y_1$ a $X \to_{S,F} Y_2$.

Z toho zrejme vyplýva, že rovnosť $\text{schéma}(S,F) = \text{schéma}(S,G)$ môže nastať aj pre rôzne množiny F a G, napríklad v predchádzajúcom príklade pre $G = F \cup \{\langle\{s_{\texttt{meno}}\},\{s_{\texttt{pohlavie}}\}\rangle\}$. Tu vidno, že funkčná závislosť $\langle\{s_{\texttt{meno}}\},\{s_{\texttt{pohlavie}}\}\rangle$ je v zmysle Armstrongovho pravidla 2a dôsledkom závislostí $\langle\{s_{\texttt{meno}}\},\{s_{\texttt{priezvisko}}\}\rangle$ a $\langle\{s_{\texttt{priezvisko}}\},\{s_{\texttt{pohlavie}}\}\rangle$ z F. Nasledujúca induktívna definícia zhŕňa všetky závislosti vzniknuté takouto aplikáciou Armstrongových pravidiel do istej množiny:

Definícia

- Nech S je množina stĺpcov s rôznymi menami a F je množina dvojíc jej podmnožín. Pod označením $\text{f-uzáver}_S(F)$ budeme rozumieť **uzáver** množiny závislostí F, t. j. najmenšiu množinu H spĺňajúcu tieto podmienky:
  - 1a Ak $Y \subseteq X$, tak $\langle X,Y\rangle \in H$.
  - 1b Ak $\langle X,Y\rangle \in F$, tak $\langle X,Y\rangle \in H$.
  - 2a Ak $\langle X,Y\rangle \in H$ a $\langle Y,Z\rangle \in H$, tak $\langle X,Z\rangle \in H$.
  - 2b1 Ak $\langle X,Y_1\rangle \in H$ a $\langle X,Y_2\rangle \in H$, tak $\langle X,Y_1 \cup Y_2\rangle \in H$.
  - 2b2 Ak $\langle X,Y_1 \cup Y_2\rangle \in H$, tak $\langle X,Y_1\rangle \in H$ a $\langle X,Y_2\rangle \in H$.

Pre naše S a F do množiny $\text{f-uzáver}_S(F)$ patria podľa bodu 1b obe funkčné závislosti z F a podľa bodu 1a všetky triviálne funkčné závislosti. Podľa bodu 2a (tranzitivity) tam potom musí patriť už spomínaná závislosť $\langle\{\text{s}_{\texttt{meno}}\},\{\text{s}_{\texttt{pohlavie}}\}\rangle$, podľa bodu 2b1 potom aj $\langle\{\text{s}_{\texttt{meno}}\},\{\text{s}_{\texttt{priezvisko}},\text{s}_{\texttt{pohlavie}}\}\rangle$. Takto pokračujeme pridávaním Armstrongovými pravidlami vynútených závislostí, až kým je množina v tomto zmysle uzavretá.

Zrejme tak (a to vo všeobecnosti) platí:

Lema

- Nech S je množina stĺpcov s rôznymi menami a F je množina dvojíc jej podmnožín. Potom platí

$$\text{schéma}(S,F) = \text{schéma}(S,\text{f-uzáver}_S(F)).$$

Otázkou je, či takto dostaneme všetky dvojice $\langle X,Y\rangle$, pre ktoré platí $X \rightarrow_{S,F} Y$. Nasledujúca lema túto intuíciu potvrdzuje:

Lema

- Nech S je množina stĺpcov s rôznymi menami a F je množina dvojíc jej podmnožín. Potom platí

$$X \rightarrow_{S,F} Y, \quad \text{akk} \quad \langle X,Y\rangle \in \text{f-uzáver}_S(F).$$

4.4.3 Pokrytie

Je intuitívne zrejmé, že čím menej funkčných závislostí (čiže akýchsi podmienok), tým menej starostí s kontrolou ich plnenia, preto sa pre danú ich množinu G oplatí hľadať čo najmenšiu množinu určujúcu tú istú schému ako G. Z predchádzajúcej state pritom vieme, že hľadáme množinu závislostí s tým istým uzáverom.

Definícia

- Nech S je množina stĺpcov s rôznymi menami a G je množina dvojíc jej podmnožín. Pod **pokrytím** množiny závislostí G rozumieme každú množinu F, pre ktorú platí $\text{f-uzáver}_S(F) = \text{f-uzáver}_S(G)$.
- Ak navyše pokrytie F obsahuje len elementárne funkčné závislosti, hovoríme o **kanonickom pokrytí**.

Ak vezmeme minule používané $S = \{\langle\texttt{meno},\text{Reťazec},0\rangle,\langle\texttt{priezvisko},\text{Reťazec},0\rangle,\langle\texttt{pohlavie},\text{Reťazec},0\rangle\}$ a $G = \{\langle\{\text{s}_{\texttt{meno}}\},\{\text{s}_{\texttt{priezvisko}}\}\rangle,\langle\{\text{s}_{\texttt{priezvisko}}\},\{\text{s}_{\texttt{pohlavie}}\}\rangle,\langle\{\text{s}_{\texttt{meno}}\},\{\text{s}_{\texttt{pohlavie}}\}\rangle\}$, pokrytiami množiny G sú napríklad spomínaná $F = \{\langle\{\text{s}_{\texttt{meno}}\},\{\text{s}_{\texttt{priezvisko}}\}\rangle,\langle\{\text{s}_{\texttt{priezvisko}}\},\{\text{s}_{\texttt{pohlavie}}\}\rangle\}$, samotná G, ale aj $H = G \cup \{\langle\{\text{s}_{\texttt{meno}},\text{s}_{\texttt{pohlavie}}\},\{\text{s}_{\texttt{meno}},\text{s}_{\texttt{pohlavie}}\}\rangle\}$ (pridali sme nič nehovoriacu triviálnu funkčnú závislosť). Pritom F a G sú kanonické, ale H nie.

Na tomto príklade vidíme, že menšie z dvoch pokrytí vzniklo odstránením závislosti, ktorá je odvoditeľná zo zvyšných dvoch. Túto myšlienku sa pokúsime zachytiť v nasledujúcej definícii:

Definícia

- Nech S je množina stĺpcov s rôznymi menami, G je množina jej závislostí (dvojíc jej podmnožín) a $f \in G$. Závislosť f nazývame **redundantnou** v G, ak sa jej odobratím z G nezmení uzáver (a teda ani príslušná schéma), t. j. keď $\text{f-uzáver}_S(G \setminus \{f\}) = \text{f-uzáver}_S(G)$.

V predchádzajúcom príklade sme videli, že v G bola redundantná závislosť $f = \langle\{\text{s}_{\texttt{meno}}\},\{\text{s}_{\texttt{pohlavie}}\}\rangle$.

Postupným odobratím všetkých takýchto nadbytočných závislostí tak dostaneme pokrytie, ktoré už žiadnu takúto redundanciu neobsahuje. Uvedomme si, že takýto postup je silne závislý na poradí odoberania redundantných závislostí, a neredundantných pokrytí preto môže byť viac. Niekedy dokonca existujú aj neredundantné pokrytia, ktoré nie sú podmnožinami pôvodnej množiny závislostí, dostaneme ich však, keď proces postupného odstraňovania nezačneme z nej, ale z jej uzáveru, ktorý je zrejme jediným maximálnym pokrytím.

Vidíme, že definícia redundantnej závislosti pracuje s uzáverom množiny daných funkčných závislostí. Ukazuje sa však, že počítaniu tohto uzáveru sa môžeme vyhnúť, a to využitím pojmu uzáveru množiny stĺpcov:

Definícia

- Nech S je množina stĺpcov s rôznymi menami, $X \subseteq S$ a F je množina závislostí na S. Pod označením $\text{s-uzáver}_{S,F}(X)$ budeme rozumieť **uzáver** množiny stĺpcov – množinu všetkých stĺpcov funkčne závislých na X, t. j.
$$\text{s-uzáver}_{S,F}(X) = \{s \in S : X \to_{S,F} \{s\}\}.$$

U nás trebárs $\text{s-uzáver}_{S,F}(\{\text{s}_{\texttt{pohlavie}}\}) = \{\text{s}_{\texttt{pohlavie}}\}$, $\text{s-uzáver}_{S,F}(\{\text{s}_{\texttt{priezvisko}}\}) = \{\text{s}_{\texttt{priezvisko}}, \text{s}_{\texttt{pohlavie}}\}$ a $\text{s-uzáver}_{S,F}(\{\text{s}_{\texttt{meno}}\}) = \text{s-uzáver}_{S,F}(\{\text{s}_{\texttt{meno}}, \text{s}_{\texttt{priezvisko}}\}) = S$.

Pojem uzáveru množiny závislých stĺpcov je užitočný preto, lebo medzi ním a uzáverom množiny závislostí možno dokázať takýto vzťah:

Lema

- Ak S je množina stĺpcov s rôznymi menami, $X, Y \subseteq S$ a F je množina závislostí na S, tak platí
$$X \to_{S,F} Y, \quad \text{akk} \quad \langle X, Y\rangle \in \text{f-uzáver}_S(F), \quad \text{akk} \quad Y \subseteq \text{s-uzáver}_{S,F}(X).$$

Ďalej sa teda stačí zaoberať uzávermi množín stĺpcov.

Princíp čo najväčšej (ale ešte postačujúcej) jednoduchosti, vo filozofii známy ako **Occamova britva**, nám opäť prikazuje zbaviť sa redundantných stĺpcov. Túto myšlienku vyjadríme v nasledujúcich definíciách:

Definícia

- Ak s je stĺpec z X taký, že
$$\text{s-uzáver}_{S,F}(X \setminus \{s\}) = \text{s-uzáver}_{S,F}(X),$$
hovoríme, že je voči X **redundantný**.
- Ak žiaden stĺpec množiny X nie je voči nej redundantný, túto množinu nazveme **redukovaná**.
- Ak je pre závislosť $\langle X, Y\rangle$ z F množina X redukovaná, hovoríme, že táto závislosť je tiež **redukovaná**.

Keďže (ako sme pred chvíľou videli) $\text{s-uzáver}_{S,F}(\{\text{s}_{\texttt{meno}}\}) = \text{s-uzáver}_{S,F}(\{\text{s}_{\texttt{meno}}, \text{s}_{\texttt{priezvisko}}\}) = S$, stĺpec $\text{s}_{\texttt{priezvisko}}$ je voči množine $\{\text{s}_{\texttt{meno}}, \text{s}_{\texttt{priezvisko}}\}$ redundantný. Táto množina teda nie je redukovaná. Naproti tomu každá jednoprvková množina redukovaná je, pretože jej uzáver obsahuje aspoň ju, ale uzáver prázdnej množiny, ktorá by z nej vznikla odobratím jej jediného stĺpca, je tiež prázdna množina, a teda neobsahuje nič. (I keď v našom príklade neexistuje viacprvková redukovaná množina, taká situácia vo všeobecnosti nie je vylúčená. Keby sme napríklad doplnili F o funkčnú závislosť $\langle\{\text{s}_{\texttt{priezvisko}}, \text{s}_{\texttt{pohlavie}}\}, \{\text{s}_{\texttt{meno}}\}\rangle$ (pomiňme jej obsahovú nezmyselnosť), redukovanou by bola aj množina $\{\text{s}_{\texttt{priezvisko}}, \text{s}_{\texttt{pohlavie}}\}$ (žiaden jej stĺpec by voči nej nebol redundantný).)

Na popis schémy by sme teda mali vyberať také pokrytie, ktoré bude kanonické a všetky závislosti v ňom redukované. V našom prípade by to bolo staré dobré $F = \{\langle\{\text{s}_{\texttt{meno}}\}, \{\text{s}_{\texttt{priezvisko}}\}\rangle, \langle\{\text{s}_{\texttt{priezvisko}}\}, \{\text{s}_{\texttt{pohlavie}}\}\rangle\}$.

4.4.4 Nadkľúč a kľúč

Všimnime si, že pre danú abstraktnú tabuľku T vzťah $\text{zhoda}(r^1, r^2, \text{M}_T)$ vlastne znamená zhodu hodnôt záznamov r^1 a r^2 vo všetkých stĺpcoch, teda inými slovami ich rovnosť. Platí teda

$$X \to_T \text{M}_T, \quad \text{akk} \quad (\forall r^1, r^2 \in \text{D}_T)\ \text{zhoda}(r^1, r^2, X) \to\ r^1 = r^2,$$

čo podozrivo pripomína vlastnosť kľúča tabuľky v SQL. Ak tento vzťah bude platiť pre všetky abstraktné tabuľky z danej schémy, pojem kľúča budeme môcť rozšíriť aj na ňu.

Definícia

- Nech S je množina stĺpcov s rôznymi menami a F je množina dvojíc jej podmnožín.
  - Množina $X \subseteq S$ bude **nadkľúč** schémy $\mathrm{schéma}(S, F)$, ak v nej platí $X \rightarrow_{S,F} S$ (t. j., ako už vieme, $\text{s-uzáver}_{S,F}(X) = S$).
  - Množina $X \subseteq S$ bude **kľúč** schémy $\mathrm{schéma}(S, F)$, ak je jej nadkľúčom, ale žiadna vlastná podmnožina množiny X nie je jej nadkľúčom.

Kľúč je teda minimálny nadkľúč. Pritom množina všetkých stĺpcov je zrejme vždy nadkľúč, z čoho vyplýva nasledujúce tvrdenie:

Lema

- Každá schéma má (aspoň jeden) kľúč.

V našom prípade pri $S = \{\langle \texttt{meno}, \mathrm{Reťazec}, 0\rangle, \langle \texttt{priezvisko}, \mathrm{Reťazec}, 0\rangle, \langle \texttt{pohlavie}, \mathrm{Reťazec}, 0\rangle\}$ a $F = \{\langle\{s_{\texttt{meno}}\}, \{s_{\texttt{priezvisko}}\}\rangle, \langle\{s_{\texttt{priezvisko}}\}, \{s_{\texttt{pohlavie}}\}\rangle\}$ je jediným kľúčom množina $\{s_{\texttt{meno}}\}$ a nadkľúčmi práve všetky jej nadmnožiny.

Kľúčov však v inom prípade môže byť aj viac. Stačí namiesto F vziať množinu funkčných závislostí $E = F \cup \{\langle\{s_{\texttt{priezvisko}}, s_{\texttt{pohlavie}}\}, \{s_{\texttt{meno}}\}\rangle\}$, a potom okrem $\{s_{\texttt{meno}}\}$ dostávame ďalší kľúč – $\{s_{\texttt{priezvisko}}, s_{\texttt{pohlavie}}\}$.

Spomedzi stĺpcov tak možno vyčleniť istú dôležitú skupinu:

Definícia

- Stĺpec s nazývame **kľúčový**, ak existuje kľúč X, ktorý ho obsahuje.

Schéma $\mathrm{schéma}(S, F)$ má teda jediný kľúčový stĺpec $s_{\texttt{meno}}$, kým kľúčovými stĺpcami schémy $\mathrm{schéma}(S, E)$ sú všetky tri.

4.4.5 1NF, 2NF, 3NF a BCNF

Všimli sme si už, že pri databázovom modelovaní nie je návrh ako návrh, no ich posudzovanie z našej strany bolo iba intuitívne. Otázka teda stojí, ako odmerať kvalitu databázového modelu. Istým (nie veľmi vydareným) pokusom pochádzajúcim už zo 70. rokov (a tento jeho vek na ňom aj vidieť) sú **normálne formy**. Existuje celá ich hierarchia, a čím vyššiu z nich navrhnutý systém (schém) tabuliek spĺňa, tým ho možno považovať za lepší.

Problematickou je hneď **prvá normálna forma** (**1NF**), ktorá vlastne ani nemá formálne vyjadrenie a navyše sa jej „definície“ často dosť líšia. Tá od každej tabuľky zo schémy okrem iného požaduje, aby jej dáta boli atomické, nedeliteľné, čiže aby nemali žiadnu vnútornú štruktúru (nezabudnuteľný priekopník databáz na východnom Slovensku kolega doktor **Honza Vinař** túto vlastnosť tabuľky nazýva výstižne „**placatosť**“). Táto požiadavka je však nezlučiteľná (napríklad) s existenciou dátového typu `DATE`, zloženého hneď z troch častí:

meno	priezvisko	dátum_narodenia
Reťazec	Reťazec	Dátum
0	0	0
Ján	Hraško	11
		7
		1987
Ružena	Šípová	1
		2
		1984
Aladár	Baba	22
		1
		1980
Ferdinand	Mravec	3
		3
		1984
Ján	Polienko	14
		4
		1982
Juraj	Truľo	16
		7
		1979
Jana	Botková	21
		9
		1977
Dana	Botková	21
		9
		1977
Ján	Hlúpy	1
		4
		1972
Aladár	Miazga	22
		12
		1987
Mikuláš	Myšiak	6
		6
		1983
Donald	Káčer	7
		10
		1982
Jozef	Námorník	23
		9
		1981
Peter	Pan	13
		1
		2001

ale trebárs i s funkciou SUBSTR, ktorá dokáže reťazcové dátové typy riadne rozpitvať. Zato ďalšia prvou normálnou formou požadovaná vlastnosť hovoriaca, že všetky riadky jednej tabuľky sú rovnako dlhé, (podľa dr. Vinařa „**ne-chlpatosť**“) je už v relačnom modeli splnená automaticky, veď čo už je toto za tabuľku:

Ján	Hraško	1
Ružena	Šípová	1
Aladár	Baba	2
Ferdinand	Mravec	3
Ján	Polienko	1
Juraj	Truľo	1
Jana	Botková	
Dana	Botková	
Ján	Hlúpy	1
Aladár	Miazga	
Mikuláš	Myšiak	4
Donald	Káčer	4
Jozef	Námorník	
Peter	Pan	5

Prvú normálnu formu však môžeme chápať ako dobre mienené upozornenie, aby sme ani len náhodou nepomysleli na takéto umiestnenie viacerých dát do jedného políčka tabuľky:

id_učiteľ	...	id_predmety
CeléČíslo	...	???
0	...	0
1	...	2, 3, 8
2	...	11, 12
3	...	7, 9
4	...	1, 6
5	...	4, 5, 10

Ďalšia normálna forma je na tom z formálneho hľadiska omnoho lepšie – tam už možno naozaj hovoriť o definícii. Aby sme však videli, čo sa pokúša vyjadriť, predstavme si takúto tabuľku:

meno	priezvisko	meniny_deň	meniny_mesiac
Reťazec	Reťazec	CeléČíslo	CeléČíslo
0	0	1	1
Ján	Hraško	24	6
Ružena	Šípová	30	8
Aladár	Baba	NULL	NULL
Ferdinand	Mravec	30	5
Ján	Polienko	24	6
Juraj	Truľo	24	4
Jana	Botková	21	8
Dana	Botková	16	4
Ján	Hlúpy	24	6
Aladár	Miazga	NULL	NULL
Mikuláš	Myšiak	6	12
Donald	Káčer	NULL	NULL
Jozef	Námorník	19	3
Peter	Pan	29	6

pričom v jej schéme predpokladáme funkčnú závislosť $\{s_{\texttt{meno}}\} \rightarrow \{s_{\texttt{meniny\_deň}}, s_{\texttt{meniny\_mesiac}}\}$ (naznačovanú, mimochodom, už názvami týchto stĺpcov). Z Armstrongových pravidiel potom máme $\{s_{\texttt{meno}}, s_{\texttt{priezvisko}}\} \rightarrow \{s_{\texttt{meniny\_deň}}, s_{\texttt{meniny\_mesiac}}\}$, z čoho aj $\{s_{\texttt{meno}}, s_{\texttt{priezvisko}}\} \rightarrow \{s_{\texttt{meno}}, s_{\texttt{priezvisko}}, s_{\texttt{meniny\_deň}}, s_{\texttt{meniny\_mesiac}}\}$. Množina $\{s_{\texttt{meno}}, s_{\texttt{priezvisko}}\}$ je teda nadkľúčom, a pretože žiadna jej podmnožina nie je nadkľúčom, ona sama je aj kľúčom. Stĺpce $s_{\texttt{meniny\_deň}}$ a $s_{\texttt{meniny\_mesiac}}$ sú teda závislé na vlastnej podmnožine kľúča, čo spôsobuje nežiadanú duplicitu dát. A naozaj, veď všetci ľudia rovnakého mena majú meniny v jeden deň, dátum menín teda (na rozdiel od dátumu narodenín) nie je atribútom toho-ktorého človeka. Schémy s takouto vlastnosťou preto treba zakázať:

Definícia

- Hovoríme, že $\text{schéma}(S, F)$ je v **druhej normálnej forme** (**2NF**), ak žiaden nekľúčový stĺpec nie je funkčne závislý na žiadnej vlastnej podmnožine žiadneho kľúča, t. j. ak neexistujú množiny $K, X \subseteq S$ a nekľúčový stĺpec s z S taký, že K je kľúč, X je jeho vlastnou podmnožinou, $s \notin X$, ale $X \rightarrow_{S,F} s$.

V predchádzajúcom príklade táto podmienka nie je splnená, stačí vziať $K = \{s_{\texttt{meno}}, s_{\texttt{priezvisko}}\}$, $X = \{s_{\texttt{meno}}\}$ a $s = s_{\texttt{meniny\_deň}}$. Schéma tejto tabuľky teda nie je v 2NF.

Uvedomme si, že keď má schéma iba jednoprvkové kľúče, podmienka v definícii sa ani pri najlepšej (vlastne najhoršej) vôli nedá porušiť, a teda takáto schéma je v 2NF automaticky. To je vlastne jeden z dôvodov, prečo pri definícii tabuľky v SQL (a teda vlastne schémy) obvykle volíme jednoprvkový kľúč.

Pokúsili sme sa takto formálne zachytiť zatiaľ len intuitívne ponímanú kvalitu návrhu schémy, no, ako vzápätí uvidíme, tento pokus má značné rezervy. Predstavme si nasledujúcu tabuľku s deklarovaným kľúčom `id` a závislosťou $\{s_{\texttt{krajina}}\} \to s_{\texttt{hlava\_krajiny}}$:

id	meno	priezvisko	krajina	hlava_krajiny
CeléČíslo	Reťazec	Reťazec	Reťazec	Reťazec
0	0	0	1	1
1	Ján	Hraško	Za siedmimi horami a siedmimi dolami	Drozdia Brada
2	Ružena	Šípová	Za siedmimi horami a siedmimi dolami	Drozdia Brada
3	Aladár	Baba	Kalifát Bagdad	Harún al-Rašíd
4	Ferdinand	Mravec	Mravenisko	Z
5	Ján	Polienko	Za siedmimi horami a siedmimi dolami	Drozdia Brada
6	Juraj	Truľo	Za siedmimi horami a siedmimi dolami	Drozdia Brada
7	Jana	Botková	NULL	NULL
8	Dana	Botková	NULL	NULL
9	Ján	Hlúpy	Za siedmimi horami a siedmimi dolami	Drozdia Brada
10	Aladár	Miazga	NULL	NULL
11	Mikuláš	Myšiak	Hollywood	Simba
12	Donald	Káčer	Hollywood	Simba
13	Jozef	Námorník	NULL	NULL
14	Peter	Pan	Neverland	NULL

Vzhľadom na to, že jediným kľúčom je jednoprvková množina $\{\texttt{id}\}$, je schéma tejto tabuľky v druhej normálnej forme, napriek tomu však problém duplicity dát nezmizol. Bude preto treba zakázať všetky netriviálne závislosti, ktorých ľavá strana nie je nadkľúč:

Definícia

- Hovoríme, že $\text{schéma}(S, F)$ je v **tretej normálnej forme** (**3NF**), ak pre každú funkčnú závislosť $X \to_{S,F} s$ platí aspoň jedna z podmienok:
  1. $s \in X$ (t. j. závislosť $X \to_{S,F} s$ je triviálna).
  2. $K \subseteq X$ pre nejaký kľúč K (t. j. X je nadkľúč).
  3. $s \in K$ pre nejaký kľúč K (t. j. stĺpec s je kľúčový).

Keďže závislosť $\{s_{\texttt{krajina}}\} \to s_{\texttt{hlava\_krajiny}}$ z predchádzajúceho príkladu nespĺňa žiadnu z týchto troch podmienok, schéma tabuľky nie je v 3NF.

Uvedomme si, že touto definíciou sme pokryli aj predchádzajúcu normálnu formu, ako je to vyjadrené v nasledujúcom tvrdení:

Lema

- Ak je schéma v tretej normálnej forme, je aj v druhej normálnej forme.

Všimnime si však ešte ďalšiu tabuľku, ktorej schéma má deklarované závislosti $\{s_{\texttt{odbor}}, s_{\texttt{zameranie}}\} \to s_{\texttt{gestor}}$ a $\{s_{\texttt{gestor}}\} \to s_{\texttt{odbor}}$:

odbor	zameranie	gestor
Reťazec	Reťazec	Reťazec
0	0	0
metamatematika	metafyzika	Premúdra
metamatematika	matika	Miazga
informatematika	matika	Vševed
informatematika	metainformatika	Vševed

Keďže závislosť $\{s_{\texttt{odbor}}, s_{\texttt{zameranie}}\} \rightarrow \{s_{\texttt{odbor}}, s_{\texttt{zameranie}}\}$ je triviálna, použitím Armstrongových pravidiel máme $\{s_{\texttt{odbor}}, s_{\texttt{zameranie}}\} \rightarrow \{s_{\texttt{odbor}}, s_{\texttt{zameranie}}, s_{\texttt{gestor}}\}$, teda množina $\{s_{\texttt{odbor}}, s_{\texttt{zameranie}}\}$ je nadkľúč. A pretože, ako vidieť z tabuľky, ani jedna z jej vlastných podmnožín nie je (nad)kľúč, ona sama je aj kľúč.

Máme tu však aj ďalší kľúč, a to $\{s_{\texttt{gestor}}, s_{\texttt{zameranie}}\}$, pretože z deklarovanej závislosti $\{s_{\texttt{gestor}}\} \rightarrow s_{\texttt{odbor}}$ máme $\{s_{\texttt{gestor}}, s_{\texttt{zameranie}}\} \rightarrow \{s_{\texttt{odbor}}\}$, čo spolu s triviálnou $\{s_{\texttt{gestor}}, s_{\texttt{zameranie}}\} \rightarrow \{s_{\texttt{zameranie}}, s_{\texttt{gestor}}\}$ dáva $\{s_{\texttt{gestor}}, s_{\texttt{zameranie}}\} \rightarrow \{s_{\texttt{odbor}}, s_{\texttt{zameranie}}, s_{\texttt{gestor}}\}$, pričom zrejme ani $s_{\texttt{gestor}}$, ani $s_{\texttt{zameranie}}$ nie sú (nad)kľúče.

Schéma našej tabuľky má teda kľúče $\{s_{\texttt{odbor}}, s_{\texttt{zameranie}}\}$ a $\{s_{\texttt{gestor}}, s_{\texttt{zameranie}}\}$. Znamená to, že všetky tri stĺpce sú kľúčové, takže schéma je v tretej normálnej forme.

Napriek tomu nám svojím spôsobom môže prekážať, že časť jedného kľúča je súčasťou druhého. Všimnime si, že závislosť $\{s_{\texttt{gestor}}\} \rightarrow s_{\texttt{odbor}}$, ktorá tento problém vlastne spôsobuje, vyhovuje iba 3. vlastnosti v definícii 3NF. Ak by sme teda povolili len prvé dve, schéma by takto upravenú definíciu nespĺňala. Takto sa dostávame k ďalšej z normálnych foriem, ktorá sa vymyká obvyklému číslovaniu:

Definícia

- Hovoríme, že $\text{schéma}(S, F)$ je v **Boyceovej-Coddovej normálnej forme** (**BCNF**), ak pre každú funkčnú závislosť $X \rightarrow_{S,F} s$ platí aspoň jedna z podmienok:
  1. $s \in X$ (t. j. závislosť $X \rightarrow_{S,F} s$ je triviálna).
  2. $K \subseteq X$ pre nejaký kľúč K (t. j. X je nadkľúč).

Znamená to, že schéma predchádzajúcej tabuľky nie je v BCNF.

Ak porovnáme posledné dve definície, bezprostredne dostávame tvrdenie, ktoré potvrdzuje očakávanú hierarchiu:

Lema

- Ak je schéma v Boyceovej-Coddovej normálnej forme, je aj v tretej normálnej forme.

Je zaujímavé, že na platnosť opačného smeru netreba tak veľa:

Lema

- Ak je schéma v tretej normálnej forme a obsahuje len jednoprvkové kľúče, je aj v Boyceovej-Coddovej normálnej forme.

Nie je preto div, že náš kontrapríklad medzi 3NF a BCNF obsahoval dvojprvkový kľúč.

4.4.6 Multizávislosti a 4NF

Keď sme v podkapitole 2.4 vytvárali návrh rozsiahlejšieho databázového systému, po krátkej chvíli sme zistili, že sa v ňom napriek všetkému nášmu snaženiu vyskytla akási nedokonalosť, prejavujúca sa v duplicitách (čiže v (spočiatku neobjavenej) súvislosti medzi dátami), a preto ho bolo treba revidovať (a to ešte nemáme istotu, že nevyvstane potreba ďalších úprav). Pritom problematika vysokej školy nám bola blízka a rozoberali sme naozaj len jej zlomok (veď čože je to – ani nie dvadsať tabuliek...). Nemožno sa preto vôbec čudovať, že návrhár, ktorý zrazu vhupne do úplne neznámej a komplexnej problematiky, neobjaví hneď na prvýkrát všetky (často rafinovane ukryté) vzťahy medzi dátami. Neraz sa nedokonalosť návrhu objaví až vo fáze, keď je už databáza v plnej paráde, a vzniknú nečakané problémy s jej udržiavaním. Vtedy si návrhár pozrie niekoľko podôb tabuľky meniacej sa v čase (čo sú vlastne abstraktné tabuľky z tej istej schémy) a snaží sa v nich nájsť doteraz neobjavenú duplicitu či závislosť. Ak nejakú nájde a usúdi, že to nie je len náhoda prejavujúca sa iba v týchto vybraných podobách tabuľky, pokúsi sa návrh zmeniť tak, aby sa v ňom už nevyskytovala. Takýchto revízií môže postupom času urobiť niekoľko (desiatok) a každá z nich je obvykle spojená s technickými

komplikáciami s prerábaním databázy. Každá nepríjemnosť tohto druhu má však aj svoju lepšiu stránku – **re-vizionára môže hriať pocit, že komplikovanej realite okolo seba porozumel o čosi viac**.

Vzťahy medzi jednotlivými faktormi sme zatiaľ vyjadrovali pomocou funkčných závislostí, no tie nemusia byť jediným zdrojom duplicity. Veď sa len pozrime na nasledujúcu tabuľku vyjadrujúcu spoločné neuzavreté predmety dvojíc našich študentov. Kvôli jednoduchosti sa obmedzme na piatakov:

predmet	študent1	študent2
Reťazec	Reťazec	Reťazec
0	0	0
Romológia	Káčer	Káčer
Romológia	Káčer	Myšiak
Romológia	Káčer	Polienko
Romológia	Myšiak	Káčer
Romológia	Myšiak	Myšiak
Romológia	Myšiak	Polienko
Romológia	Polienko	Káčer
Romológia	Polienko	Myšiak
Romológia	Polienko	Polienko
Teória hier	Káčer	Káčer
Teória hier	Káčer	Polienko
Teória hier	Polienko	Káčer
Teória hier	Polienko	Polienko

Ľahko sa môžeme presvedčiť, že žiadna z dvojprvkových (a teda ani jednoprvkových) podmnožín množiny všetkých stĺpcov $\{\mathrm{s}_{\texttt{predmet}}, \mathrm{s}_{\texttt{študent1}}, \mathrm{s}_{\texttt{študent2}}\}$ nie je nadkľúč, preto je jediným kľúčom schémy tejto tabuľky práve táto množina. Zároveň vidieť, že táto tabuľka (a teda aj jej schéma) odporuje všetkým potenciálnym netriviálnym závislostiam, čo znamená, že celá schéma je v Boyceovej-Coddovej normálnej forme. Napriek tomu v oboch častiach tabuľky oddelených dvojčiarou možno pozorovať istú duplicitu – dvojice vzniknuté z príslušných riadkov obmedzených na stĺpce $\mathrm{s}_{\texttt{študent1}}$ a $\mathrm{s}_{\texttt{študent2}}$ sú vlastne všetkými kombináciami prvkov istých množín (dokonca, v tomto prípade tej istej). Navyše, hrozia problémy s údržbou – ak by napríklad Káčer úspešne uzavrel predmet Romológia, bolo by treba z tabuľky odstraňovať nie jeden, ale hneď päť záznamov. Vidíme teda, že ani pojem závislosti, ani slávna Boyceova-Coddova normálna forma tu nepomôžu. Uvedený typ duplicity sa preto pokúsime vyjadriť iným pojmom:

Definícia

- Nech T je abstraktná tabuľka a nech X a Y sú podmnožiny množiny jej stĺpcov. Potom hovoríme, že množina X **multiurčuje** množinu Y alebo (ekvivalentne) že Y je **multizávislá** na X, a píšeme $X \twoheadrightarrow_T Y$, ak pre všetky dvojice záznamov r^1 a r^2 zhodujúce sa na množine X existujú dva záznamy p^1 a p^2 také, že oba sa zhodujú s oboma r^1 a r^2 na množine X, p^1 sa zhoduje s r^1 na Y a s r^2 na $\mathrm{M}_T \setminus (X \cup Y)$, a naopak, p^2 sa zhoduje s r^2 na Y a s r^1 na $\mathrm{M}_T \setminus (X \cup Y)$. Formálne teda

$$\begin{aligned} X \twoheadrightarrow_T Y, \quad \text{akk} \quad & (\forall r^1, r^2 \in \mathrm{D}_T)\ \mathrm{zhoda}(r^1, r^2, X) \to ((\exists p^1, p^2 \in \mathrm{D}_T) \\ & (\mathrm{zhoda}(p^1, r^1, X \cup Y) \wedge \mathrm{zhoda}(p^1, r^2, \mathrm{M}_T \setminus (X \cup Y)) \wedge \\ & \wedge\, \mathrm{zhoda}(p^2, r^2, X \cup Y) \wedge \mathrm{zhoda}(p^2, r^1, \mathrm{M}_T \setminus (X \cup Y)))). \end{aligned}$$

  Dvojicu potom $\langle X, Y \rangle$ nazývame **multizávislosť**.
- Ak platí $Y \subseteq X$, hovoríme o **triviálnej multizávislosti**.

Všimnime si, že v predchádzajúcom príklade máme netriviálnu multizávislosť $\{\mathrm{s}_{\texttt{predmet}}\} \twoheadrightarrow \{\mathrm{s}_{\texttt{študent1}}\}$. Pre ilustráciu toho, že podmienka z definície je pre množiny stĺpcov $X = \{\mathrm{s}_{\texttt{predmet}}\}$ a $Y = \{\mathrm{s}_{\texttt{študent1}}\}$ splnená,

ju ukážme aspoň pre jednu dvojicu záznamov. Ak r^1 je 5. riadok (vieme, pravdaže, že o poradí riadkov nemá pri abstraktných tabuľkách zmysel hovoriť, ide nám tu o poradie, v ktorom sú napísané), t. j.

$$r^1 = \{\langle \mathrm{s}_{\texttt{predmet}}, \texttt{Romológia}\rangle, \langle \mathrm{s}_{\texttt{študent1}}, \texttt{Myšiak}\rangle, \langle \mathrm{s}_{\texttt{študent2}}, \texttt{Myšiak}\rangle\},$$

a r^2 je 7. riadok, t. j.

$$r^2 = \{\langle \mathrm{s}_{\texttt{predmet}}, \texttt{Romológia}\rangle, \langle \mathrm{s}_{\texttt{študent1}}, \texttt{Polienko}\rangle, \langle \mathrm{s}_{\texttt{študent2}}, \texttt{Káčer}\rangle\},$$

ktoré sa zhodujú na množine $X = \{\mathrm{s}_{\texttt{predmet}}\}$, stačí vziať ako p^1 4. riadok, t. j.

$$p^1 = \{\langle \mathrm{s}_{\texttt{predmet}}, \texttt{Romológia}\rangle, \langle \mathrm{s}_{\texttt{študent1}}, \texttt{Myšiak}\rangle, \langle \mathrm{s}_{\texttt{študent2}}, \texttt{Káčer}\rangle\},$$

a ako p^2 8. riadok, t. j.

$$p^2 = \{\langle \mathrm{s}_{\texttt{predmet}}, \texttt{Romológia}\rangle, \langle \mathrm{s}_{\texttt{študent1}}, \texttt{Polienko}\rangle, \langle \mathrm{s}_{\texttt{študent2}}, \texttt{Myšiak}\rangle\}$$

(takže hodnoty v jedinom stĺpci $\mathrm{s}_{\texttt{študent2}}$ z množiny $\mathrm{M}_T \setminus (X \cup Y)$ sú oproti r^1 a r^2 vymenené). A naozaj, potom platia všetky štyri vzťahy $\mathrm{zhoda}(p^1, r^1, X \cup Y)$, $\mathrm{zhoda}(p^1, r^2, \mathrm{M}_T \setminus (X \cup Y))$, $\mathrm{zhoda}(p^2, r^2, X \cup Y)$ aj $\mathrm{zhoda}(p^2, r^1, \mathrm{M}_T \setminus (X \cup Y))$.

Ak si našu tabuľku predstavíme s navzájom vymeneným druhým a tretím stĺpcom, okamžite nás napadne, že okrem $\{\mathrm{s}_{\texttt{predmet}}\} \twoheadrightarrow \{\mathrm{s}_{\texttt{študent1}}\}$ tu máme aj symetrickú multizávislosť $\{\mathrm{s}_{\texttt{predmet}}\} \twoheadrightarrow \{\mathrm{s}_{\texttt{študent2}}\}$, len sme namiesto množiny Y vzali množinu $\mathrm{M}_T \setminus (X \cup Y)$. Z definície ľahko vidieť, že tento obrat je platný vo všeobecnosti:

Lema

- Nech T je abstraktná tabuľka a nech X a Y sú podmnožiny množiny jej stĺpcov. Potom ak platí $X \twoheadrightarrow_T Y$, tak platí aj $X \twoheadrightarrow_T (\mathrm{M}_T \setminus (X \cup Y))$.

Práve týmto netriviálnym multizávislostiam by sme sa chceli vyhnúť, čo vyjadríme ďalšou normálnou formou. Za povšimnutie stojí, že tentoraz ju definujeme len pre tabuľku, a nie pre celú schému, veď pojmy závislosti (pomocou ktorého je schéma definovaná) a multizávislosti spolu vlastne vôbec nesúvisia:

Definícia

- Hovoríme, že tabuľka T je v **štvrtej normálnej forme** (**4NF**), ak je každá jej multizávislosť netriviálna.

Keďže naša tabuľka takúto netriviálnu multizávislosť obsahuje, nie je v 4NF.

4.4.7 Prirodzené spojenie, projekcia a bezstratová dekompozícia

Ukázali sme, s akými všelijakými zádrheľmi sa môže návrhár databázy postretnúť. Vzniká však prirodzená otázka, ako sa s nimi, keď sa už vyskytnú, vysporiadať. Aby sme dôkladnejšie rozobrali túto nepríjemnú situáciu, budeme potrebovať nasledujúce pojmy. Prvý z nich nám nie je úplne neznámy, pretože sme sa už stretli s jeho miernou obmenou pre typované relácie – hovorili sme jej spojenie.

Definícia

- Nech T^1 a T^2 sú abstraktné tabuľky s metadátami $\mathrm{M}_{T^1} = X^1 \cup Y^1$, resp. $\mathrm{M}_{T^2} = X^2 \cup Y^2$, kde $\langle X^1, Y^1\rangle$, $\langle X^2, Y^2\rangle$ a $\langle Y^1, Y^2\rangle$ sú dvojice navzájom disjunktných množín stĺpcov, ďalej $X^1 = \{s_1^1, \ldots, s_n^1\}$, $X^2 = \{s_1^2, \ldots, s_n^2\}$ a pre všetky $i \in \{1, \ldots, n\}$ platí $s_i^1.\mathrm{názov} = s_i^2.\mathrm{názov}$ a $s_i^1.\mathrm{typ} = s_i^2.\mathrm{typ}$ (t. j. nevyžadujeme $s_i^1.\mathrm{nullabilita} = s_i^2.\mathrm{nullabilita}$). Označme (pre každé $i \in \{1, \ldots, n\}$) s_i stĺpec taký, že platí:
  - $s_i.\mathrm{názov} = s_i^1.\mathrm{názov} = s_i^2.\mathrm{názov}$,
  - $s_i.\mathrm{typ} = s_i^1.\mathrm{typ} = s_i^2.\mathrm{typ}$,

- $s_i.\text{nullabilita} = \min\{s_i^1.\text{nullabilita}, s_i^2.\text{nullabilita}\}$.

Potom **prirodzeným spojením** tabuliek T^1 a T^2 budeme nazývať abstraktnú tabuľku T s metadátami $\text{M}_T = \{s_1, \ldots, s_n\} \cup Y^1 \cup Y^2$ a dátami

$$\text{D}_T = \{r : (\exists r^1 \in \text{D}_{T^1})(\exists r^2 \in \text{D}_{T^2})\ \text{zhoda}(r, r^1, Y^1) \wedge \text{zhoda}(r, r^2, Y^2) \wedge$$
$$\wedge\ (\forall i \in \{1, \ldots, n\}) r(s_i) = r^1(s_i^1) = r^2(s_i^2)\}$$

a budeme ho označovať $T^1 \bowtie T^2$.

Všimnime si, že ak je nullabilita všetkých stĺpcov z X^1 a X^2 nulová, znamená to vlastne, že $X^1 = X^2$ a že $s_i = s_i^1 = s_i^2$, a potom pri označení $X = X^1 = X^2$ máme zjednodušene $\text{M}_{T^1 \bowtie T^2} = X \cup Y^1 \cup Y^2$ a

$$\text{D}_{T^1 \bowtie T^2} = \{r : (\exists r^1 \in \text{D}_{T^1})(\exists r^2 \in \text{D}_{T^2})\ \text{zhoda}(r, r^1, X \cup Y^1) \wedge \text{zhoda}(r, r^2, X \cup Y^2)\}.$$

V prípade abstraktných tabuliek T^1 a T^2:

meno	priezvisko	krajina
Reťazec	Reťazec	Reťazec
0	0	1
Ján	Hraško	Za siedmimi horami a siedmimi dolami
Ružena	Šípová	Za siedmimi horami a siedmimi dolami
Aladár	Baba	Kalifát Bagdad
Ferdinand	Mravec	Mravenisko
Ján	Polienko	Za siedmimi horami a siedmimi dolami
Juraj	Truľo	Za siedmimi horami a siedmimi dolami
Jana	Botková	NULL
Dana	Botková	NULL
Ján	Hlúpy	Za siedmimi horami a siedmimi dolami
Aladár	Miazga	NULL
Mikuláš	Myšiak	Hollywood
Donald	Káčer	Hollywood
Jozef	Námorník	NULL
Peter	Pan	Neverland

krajina	hlava
Reťazec	Reťazec
0	1
Za siedmimi horami a siedmimi dolami	Drozdia Brada
Kalifát Bagdad	Harún al-Rašíd
Mravenisko	Z
Hollywood	Simba
Neverland	NULL
Haliganda	NULL

máme mohutnosti oboch množín spájaných stĺpcov $n = 1$, tieto stĺpce sú $s_1^1 = \langle \texttt{krajina}, \text{Reťazec}, 1\rangle$ a $s_1^2 = \langle \texttt{krajina}, \text{Reťazec}, 0\rangle$ (vidíme, že sa líšia len v nullabilite). Množiny ostatných stĺpcov sú $Y^1 = \{\langle \texttt{meno}, \text{Reťazec}, 0\rangle, \langle \texttt{priezvisko}, \text{Reťazec}, 0\rangle\}$ v T^1 a $Y^2 = \{\langle \texttt{hlava}, \text{Reťazec}, 1\rangle\}$ v T^2. Podľa definície potom $s_1 = \langle \texttt{krajina}, \text{Reťazec}, 0\rangle$ (lebo $s_1.\text{nullabilita} = \min\{s_1^1.\text{nullabilita}, s_1^2.\text{nullabilita}\} = \min\{1, 0\} = 0$), takže prirodzené spojenie $T^1 \bowtie T^2$ týchto tabuliek je abstraktná tabuľka:

meno	priezvisko	krajina	hlava
Reťazec	Reťazec	Reťazec	Reťazec
0	0	0	1
Ján	Hraško	Za siedmimi horami a siedmimi dolami	Drozdia Brada
Ružena	Šípová	Za siedmimi horami a siedmimi dolami	Drozdia Brada
Aladár	Baba	Kalifát Bagdad	Harún al-Rašíd
Ferdinand	Mravec	Mravenisko	Z
Ján	Polienko	Za siedmimi horami a siedmimi dolami	Drozdia Brada
Juraj	Truľo	Za siedmimi horami a siedmimi dolami	Drozdia Brada
Ján	Hlúpy	Za siedmimi horami a siedmimi dolami	Drozdia Brada
Mikuláš	Myšiak	Hollywood	Simba
Donald	Káčer	Hollywood	Simba
Peter	Pan	Neverland	NULL

Všimnime si, že na rozdiel od spojenia typovaných relácií, kde sme požadovali rôznosť mien všetkých stĺpcov, tu sú stĺpce s rovnakými menami priam nutné – prirodzené spojenie sa deje práve cez ne. Ďalším rozdielom je práca s prázdnou hodnotou – tu sa k nej správame ako k ľubovoľnej inej, nerobíme pri nej žiadnu výnimku.

V istom zmysle opačným procesom k spojeniu je už pri typovaných reláciách spomínaná projekcia. Definujme ju aj pre abstraktné tabuľky:

Definícia

- Nech T je abstraktná tabuľka a nech $X \subseteq \mathrm{M}_T$. Potom **projekciou** tabuľky T na množinu X budeme nazývať abstraktnú tabuľku P s metadátami $\mathrm{M}_P = X$ a dátami

$$\mathrm{D}_P = \{p : (\exists r \in \mathrm{D}_T)\ \mathrm{zhoda}(p, r, X)\}$$

  a označovať ju budeme $\pi(T, X)$.

Ak urobíme projekciu predchádzajúceho výsledného prirodzeného spojenia na stĺpce s názvami `krajina` a `hlava`, dostávame abstraktnú tabuľku P^2:

`krajina`	`hlava`
Reťazec	Reťazec
0	1
`Za siedmimi horami a siedmimi dolami`	`Drozdia Brada`
`Kalifát Bagdad`	`Harún al-Rašíd`
`Mravenisko`	`Z`
`Hollywood`	`Simba`
`Neverland`	`NULL`

Od pôvodnej tabuľky T^2 sa táto P^2 líši len chýbajúcim záznamom Haligandy, a to práve preto, že ten sa v prirodzenom spojení neangažoval. Ak by tento záznam v T^2 nebol, výsledok tejto dvojoperácie (príslušné prirodzené spojenie a následná projekcia) by sa vôbec nezmenil.

Projekcia P^1 na stĺpce s názvami `meno`, `priezvisko` a `krajina` vyzerá takto:

`meno`	`priezvisko`	`krajina`
Reťazec	Reťazec	Reťazec
0	0	0
`Ján`	`Hraško`	`Za siedmimi horami a siedmimi dolami`
`Ružena`	`Šípová`	`Za siedmimi horami a siedmimi dolami`
`Aladár`	`Baba`	`Kalifát Bagdad`
`Ferdinand`	`Mravec`	`Mravenisko`
`Ján`	`Polienko`	`Za siedmimi horami a siedmimi dolami`
`Juraj`	`Truľo`	`Za siedmimi horami a siedmimi dolami`
`Ján`	`Hlúpy`	`Za siedmimi horami a siedmimi dolami`
`Mikuláš`	`Myšiak`	`Hollywood`
`Donald`	`Káčer`	`Hollywood`
`Peter`	`Pan`	`Neverland`

Aj tu vidíme rozdiely medzi P^1 a pôvodnou tabuľkou T^1 – chýbajú riadky s prázdnymi hodnotami (nebolo ich na čo napojiť) a líšia sa aj nullability stĺpcov s názvom `krajina`. Ak by sme však štartovali nie z T^1, ale z P^1, výsledok tejto dvojoperácie by bol opäť P^1.

Črtá sa teda takéto tvrdenie:

Lema

- Nech X, Y^1 a Y^2 sú disjunktné množiny stĺpcov s navzájom rôznymi názvami. Nech T^1 a T^2 sú abstraktné tabuľky s metadátami $\mathrm{M}_{T^1} = X \cup Y^1$, resp. $\mathrm{M}_{T^2} = X \cup Y^2$. Navyše nech $\pi(T^1, X) =$

$\pi(T^2, X)$ (t. j. všetky riadky oboch tabuliek sú spojiteľné aspoň s jedným riadkom tej druhej tabuľky). Potom platí

$$\pi(T^1 \bowtie T^2, X \cup Y^1) = T^1$$

(a, samozrejme, symetricky aj $\pi(T^1 \bowtie T^2, X \cup Y^2) = T^2$).

Zaujímavé je to aj s opačným postupom, keď najprv urobíme dve projekcie a potom urobíme ich prirodzené spojenie. Ani tu sa totiž nemusíme vrátiť k pôvodnej tabuľke, ako ukazuje nasledujúci príklad:

predmet	študent1	študent2
Reťazec	Reťazec	Reťazec
0	0	0
Romológia	Káčer	Káčer
Romológia	Myšiak	Myšiak
Romológia	Polienko	Polienko
Teória hier	Káčer	Káčer
Teória hier	Polienko	Polienko

Ak túto tabuľku označíme T a vezmeme $X = \{s_{\texttt{predmet}}\}$, $Y^1 = \{s_{\texttt{študent1}}\}$ a $Y^2 = \{s_{\texttt{študent2}}\}$, kompozícia (takmer totožných) projekcií na $X \cup Y^1$ i na $X \cup Y^2$:

predmet	študent1
Reťazec	Reťazec
0	0
Romológia	Káčer
Romológia	Myšiak
Romológia	Polienko
Teória hier	Káčer
Teória hier	Polienko

$\bowtie$

predmet	študent2
Reťazec	Reťazec
0	0
Romológia	Káčer
Romológia	Myšiak
Romológia	Polienko
Teória hier	Káčer
Teória hier	Polienko

je naša známa tabuľka:

predmet	študent1	študent2
Reťazec	Reťazec	Reťazec
0	0	0
Romológia	Káčer	Káčer
Romológia	Káčer	Myšiak
Romológia	Káčer	Polienko
Romológia	Myšiak	Káčer
Romológia	Myšiak	Myšiak
Romológia	Myšiak	Polienko
Romológia	Polienko	Káčer
Romológia	Polienko	Myšiak
Romológia	Polienko	Polienko
Teória hier	Káčer	Káčer
Teória hier	Káčer	Polienko
Teória hier	Polienko	Káčer
Teória hier	Polienko	Polienko

Výsledok sa teda naozaj líši od tabuľky, z ktorej sme štartovali. Jej dáta sú však aspoň nadmnožinou dát pôvodnej tabuľky. Vo všeobecnosti totiž platí:

Lema

- Nech T je abstraktná tabuľka a nech X, Y^1 a Y^2 sú disjunktné množiny stĺpcov také, že $\mathrm{M}_T = X \cup Y^1 \cup Y^2$. Ak označíme

$$P = \pi(T, X \cup Y^1) \bowtie \pi(T, X \cup Y^2),$$

  tak platí $\mathrm{M}_P = \mathrm{M}_T$ a $\mathrm{D}_P \supseteq \mathrm{D}_T$.

Prípadná rovnosť medzi pôvodnými a výslednými dátami je taká dôležitá, že si zaslúži osobitnú definíciu:

Definícia

- Nech T je abstraktná tabuľka a nech X, Y^1 a Y^2 sú disjunktné množiny stĺpcov také, že $\mathrm{M}_T = X \cup Y^1 \cup Y^2$. Ak platí

$$T = \pi(T, X \cup Y^1) \bowtie \pi(T, X \cup Y^2),$$

  tak dvojicu T^1 a T^2 nazveme **bezstratovou dekompozíciou** abstraktnej tabuľky T.

Uvedomme si, že tento pojmový aparát nám umožňuje formulovať užitočnú a jednoduchú podmienku ekvivalentnú s definíciou 4NF:

Lema

- Abstraktná tabuľka T je v 4NF práve vtedy, keď M_T nemožno bezo zvyšku rozložiť na disjunktné množiny stĺpcov X, Y^1 a Y^2 také, že T sa dá bezstratovo dekomponovať na svoje projekcie na množiny $X \cup Y^1$ a $X \cup Y^2$.

Ľahko vidieť, že v prípade našej problematickej tabuľky:

predmet	študent1	študent2
Reťazec	Reťazec	Reťazec
0	0	0
Romológia	Káčer	Káčer
Romológia	Káčer	Myšiak
Romológia	Káčer	Polienko
Romológia	Myšiak	Káčer
Romológia	Myšiak	Myšiak
Romológia	Myšiak	Polienko
Romológia	Polienko	Káčer
Romológia	Polienko	Myšiak
Romológia	Polienko	Polienko
Teória hier	Káčer	Káčer
Teória hier	Káčer	Polienko
Teória hier	Polienko	Káčer
Teória hier	Polienko	Polienko

takúto bezstratovú dekompozíciu urobiť naozaj možno – opäť stačí vziať $X = \{\mathrm{s}_{\texttt{predmet}}\}$, $Y^1 = \{\mathrm{s}_{\texttt{študent1}}\}$ a $Y^2 = \{\mathrm{s}_{\texttt{študent2}}\}$ (a asi neprekvapí, že tieto množiny zodpovedajú multizávislostiam). Bezstratová dekompozícia je potom:

predmet	študent1
Reťazec	Reťazec
0	0
Romológia	Káčer
Romológia	Myšiak
Romológia	Polienko
Teória hier	Káčer
Teória hier	Polienko

⋈

predmet	študent2
Reťazec	Reťazec
0	0
Romológia	Káčer
Romológia	Myšiak
Romológia	Polienko
Teória hier	Káčer
Teória hier	Polienko

Za zmienku stojí, že obe tabuľky sú (až na drobný detail v názve jedného stĺpca) zhodné, čo však vzhľadom na to, že v oboch prípadoch ide o predmety ešte chýbajúce piatakom, nie je vôbec prekvapivé. Ich prirodzeným spojením zrejme naozaj dostaneme všetky dvojice piatakov so spoločným zapísaným, ale ešte neuzavretým predmetom, dekompozícia je teda naozaj zmysluplná. (Mimochodom, takéto záverečné zhodnotenie dekompozície je vždy užitočné, je totiž zároveň jej sémantickou kontrolou.)

Všimnime si, že vo výsledných tabuľkách už žiaden problém s duplicitou nemáme. Vo všeobecnosti sa však môže stať, že niektorá z projekcií (alebo hoci obe) ešte obsahuje multizávislosti. V takom prípade proces dekompozície iterujeme, až kým sa problému nezbavíme.

4.4.8 Dekompozícia schém

Nechajme teraz piatacké resty, veď sme ich vcelku uspokojivo vyriešili, a všimnime si radšej svoje. Ukážeme, že aj tu nám pojem bezstratovej dekompozície výrazne pomôže.

Najprv pripomeňme tabuľku:

meno	priezvisko	meniny_deň	meniny_mesiac
Reťazec	Reťazec	CeléČíslo	CeléČíslo
0	0	1	1
Ján	Hraško	24	6
Ružena	Šípová	30	8
Aladár	Baba	NULL	NULL
Ferdinand	Mravec	30	5
Ján	Polienko	24	6
Juraj	Truľo	24	4
Jana	Botková	21	8
Dana	Botková	16	4
Ján	Hlúpy	24	6
Aladár	Miazga	NULL	NULL
Mikuláš	Myšiak	6	12
Donald	Káčer	NULL	NULL
Jozef	Námorník	19	3
Peter	Pan	29	6

ktorej schéma nebola ani len v 2NF, lebo obsahovala funkčnú závislosť $\{s_{\texttt{meno}}\} \rightarrow \{s_{\texttt{meniny\_deň}}, s_{\texttt{meniny\_mesiac}}\}$, ale jej kľúč nebol $\{s_{\texttt{meno}}\}$, ale až $\{s_{\texttt{meno}}, s_{\texttt{priezvisko}}\}$. Tu sa nám núka dekompozícia na takéto dve projekcie:

meno	priezvisko
Reťazec	Reťazec
0	0
Ján	Hraško
Ružena	Šípová
Aladár	Baba
Ferdinand	Mravec
Ján	Polienko
Juraj	Truľo
Jana	Botková
Dana	Botková
Ján	Hlúpy
Aladár	Miazga
Mikuláš	Myšiak
Donald	Káčer
Jozef	Námorník
Peter	Pan

$\bowtie$

meno	meniny_deň	meniny_mesiac
Reťazec	CeléČíslo	CeléČíslo
0	1	1
Ján	24	6
Ružena	30	8
Aladár	NULL	NULL
Ferdinand	30	5
Juraj	24	4
Jana	21	8
Dana	16	4
Mikuláš	6	12
Donald	NULL	NULL
Jozef	19	3
Peter	29	6

Ľahko vidieť, že táto dekompozícia je v tomto prípade naozaj bezstratová. Bude to tak však pre ľubovoľnú tabuľku z tejto schémy? A ak áno, bude možné niečo povedať o schémach oboch výsledných projekcií? Aby sme tieto otázky vedeli zodpovedať, definujme pojem projekcie pre schémy (a popritom aj pre funkčné závislosti):

Definícia

- Nech S je množina stĺpcov s rôznymi menami, F je množina ich závislostí a U je podmnožina S.
  - **Projekciou množiny závislostí** F na U budeme nazývať množinu závislostí

$$\widetilde{\pi}(S,F,U) = \{\langle X,Y\rangle \in \text{f-uzáver}_S(F) : X \cup Y \subseteq U \wedge X \cap Y = \emptyset\}.$$

  - **Projekciou schémy** $\text{schéma}(S,F)$ na U budeme rozumieť schému $\text{schéma}(U,\widetilde{\pi}(S,F,U))$.

Napríklad ak $S = \{s_1,s_2,s_3\}$, $F = \{\langle\{s_1\},\{s_2\}\rangle,\langle\{s_2\},\{s_3\}\rangle\}$ a $U = \{s_1,s_3\}$, tak projekcia množiny závislostí F na U je $\{\langle\{s_1\},\{s_3\}\rangle\}$. Projekcia $\widetilde{\pi}(S,F,U)$ schémy $\text{schéma}(S,F)$ na toto U teda obsahuje jedinú netriviálnu funkčnú závislosť $\{s_1\} \rightarrow s_3$. Zároveň si všimnime, že by bolo chybou v definícii projekcie množiny závislostí namiesto $\text{f-uzáver}_S(F)$ zobrať iba F, v takom prípade by sme totiž chybne dostali $\widetilde{\pi}(S,F,U) = \emptyset$. Projekcia schémy by totiž, ako napovedá jej názov, mala obsahovať práve projekcie jednotlivých tabuliek z tejto schémy. Presne to tvrdí nasledujúca lema:

Lema

- Nech S je množina stĺpcov s rôznymi menami, F je množina ich závislostí a U je podmnožina S. Potom

$$\text{schéma}(U,\widetilde{\pi}(S,F,U)) = \{\pi(T,U) : T \in \text{schéma}(S,F)\}.$$

Ako sme videli v predošlej stati, svojím spôsobom opačným procesom k projekcii je spojenie. Tu ho definujeme pre schémy:

Definícia

- Nech pre $i \in \{1,2\}$ je S^i množina stĺpcov s rôznymi menami a F^i je množina ich funkčných závislostí. **Spojením schém** $\text{schéma}(S^1,F^1)$ a $\text{schéma}(S^2,F^2)$ nazývame schému

$$\text{schéma}(S^1,F^1) \bowtie \text{schéma}(S^2,F^2) = \text{schéma}(S^1 \cup S^2, F^1 \cup F^2).$$

Aj tu sa dá očakávať, že tabuľky z výslednej schémy sú práve spojeniami tabuliek z jednotlivých faktorov:

Lema

- Nech pre $i \in \{1, 2\}$ je S^i množina stĺpcov s rôznymi menami a F^i je množina ich závislostí. Potom

$$\text{schéma}(S^1, F^1) \bowtie \text{schéma}(S^2, F^2) = \{T^1 \bowtie T^2 : T^1 \in \text{schéma}(S^1, F^1) \wedge T^2 \in \text{schéma}(S^2, F^2)\}.$$

Zovšeobecňovanie pojmov z tabuliek na schémy zakončíme pri dekompozícii:

Definícia

- Nech S je množina stĺpcov s rôznymi menami a F je množina ich závislostí. Nech X, Y^1 a Y^2 sú disjunktné množiny stĺpcov také, že $S = X \cup Y^1 \cup Y^2$. Ak platí

$$\text{schéma}(S, F) = \widetilde{\pi}(S, F, X \cup Y^1) \bowtie \widetilde{\pi}(S, F, X \cup Y^2),$$

dvojicu schém $\widetilde{\pi}(S, F, X \cup Y^1)$ a $\widetilde{\pi}(S, F, X \cup Y^2)$ nazveme **bezstratovou dekompozíciou schémy** $\text{schéma}(S, F)$.

A teraz už môžeme vysloviť dôležitú vetu, ktorá odpovie na horeuvedené otázky:

Veta

- Nech S je množina stĺpcov s rôznymi menami a F je množina ich závislostí. Nech X, Y^1 a Y^2 sú disjunktné množiny stĺpcov také, že $S = X \cup Y^1 \cup Y^2$. Potom platí, že dekompozícia $\text{schéma}(S, F)$ na schémy $\widetilde{\pi}(S, F, X \cup Y^1)$ a $\widetilde{\pi}(S, F, X \cup Y^2)$ je bezstratová práve vtedy, keď platí $X \rightarrow_{S,F} Y^1$ alebo $X \rightarrow_{S,F} Y^2$.

Teraz už vidíme, že navrhnutá dekompozícia tabuľky z predchádzajúceho príkladu má všeobecnejšiu platnosť – možno ju urobiť pre ľubovoľnú tabuľku jej schémy

$$\text{schéma}(\{\text{s}_{\texttt{meno}}, \text{s}_{\texttt{priezvisko}}, \text{s}_{\texttt{meniny\_deň}}, \text{s}_{\texttt{meniny\_mesiac}}\}, \{\langle\{\text{s}_{\texttt{meno}}\}, \{\text{s}_{\texttt{meniny\_deň}}, \text{s}_{\texttt{meniny\_mesiac}}\}\rangle\}),$$

pretože tú možno podľa tejto vety bezstratovo dekomponovať na dvojicu schém

$$\text{schéma}(\{\text{s}_{\texttt{meno}}, \text{s}_{\texttt{priezvisko}}\}, \emptyset)$$

a

$$\text{schéma}(\{\text{s}_{\texttt{meno}}, \text{s}_{\texttt{meniny\_deň}}, \text{s}_{\texttt{meniny\_mesiac}}\}, \{\langle\{\text{s}_{\texttt{meno}}\}, \{\text{s}_{\texttt{meniny\_deň}}, \text{s}_{\texttt{meniny\_mesiac}}\}\rangle\}).$$

Pri prvom probléme sme sa dosť zdržali, no tam vybudovaný aparát nám riešenie zvyšných dvoch problémov značne urýchli. Pozrime sa na ďalšiu problematickú tabuľku:

id	meno	priezvisko	krajina	hlava_krajiny
CeléČíslo	Reťazec	Reťazec	Reťazec	Reťazec
0	0	0	1	1
1	Ján	Hraško	Za siedmimi horami a siedmimi dolami	Drozdia Brada
2	Ružena	Šípová	Za siedmimi horami a siedmimi dolami	Drozdia Brada
3	Aladár	Baba	Kalifát Bagdad	Harún al-Rašíd
4	Ferdinand	Mravec	Mravenisko	Z
5	Ján	Polienko	Za siedmimi horami a siedmimi dolami	Drozdia Brada
6	Juraj	Truľo	Za siedmimi horami a siedmimi dolami	Drozdia Brada
7	Jana	Botková	NULL	NULL
8	Dana	Botková	NULL	NULL
9	Ján	Hlúpy	Za siedmimi horami a siedmimi dolami	Drozdia Brada
10	Aladár	Miazga	NULL	NULL
11	Mikuláš	Myšiak	Hollywood	Simba
12	Donald	Káčer	Hollywood	Simba
13	Jozef	Námorník	NULL	NULL
14	Peter	Pan	Neverland	NULL

ktorej schéma

$$\text{schéma}(\{s_{\texttt{id}}, s_{\texttt{meno}}, s_{\texttt{priezvisko}}, s_{\texttt{krajina}}, s_{\texttt{hlava\_krajiny}}\},$$
$$\{\langle\{s_{\texttt{id}}\}, \{s_{\texttt{meno}}, s_{\texttt{priezvisko}}, s_{\texttt{krajina}}, s_{\texttt{hlava\_krajiny}}\}\rangle, \langle\{s_{\texttt{krajina}}\}, \{s_{\texttt{hlava\_krajiny}}\}\rangle\}\})$$

je v 2NF, ale nie v 3NF. Podľa predchádzajúcej vety pre $X = \{s_{\texttt{krajina}}\}$, $Y^1 = \{s_{\texttt{id}}, s_{\texttt{meno}}, s_{\texttt{priezvisko}}\}$ a $Y^2 = \{s_{\texttt{hlava\_krajiny}}\}$ dostávame bezstratovú dekompozíciu na schémy

$$\text{schéma}(\{s_{\texttt{id}}, s_{\texttt{meno}}, s_{\texttt{priezvisko}}, s_{\texttt{krajina}}\}, \{\langle\{s_{\texttt{id}}\}, \{s_{\texttt{meno}}, s_{\texttt{priezvisko}}, s_{\texttt{krajina}}\}\rangle\})$$

a

$$\text{schéma}(\{s_{\texttt{krajina}}, s_{\texttt{hlava\_krajiny}}\}, \{\langle\{s_{\texttt{krajina}}\}, \{s_{\texttt{hlava\_krajiny}}\}\rangle\}).$$

Konkrétne naša tabuľka bude dekomponovaná na dvojicu (za zmienku stojí neobvyklý riadok druhej tabuľky s oboma hodnotami prázdnymi, je to však v súlade s definíciou projekcie i prirodzeného spojenia):

id	meno	priezvisko	krajina
CeléČíslo	Reťazec	Reťazec	Reťazec
0	0	0	1
1	Ján	Hraško	Za siedmimi horami a siedmimi dolami
2	Ružena	Šípová	Za siedmimi horami a siedmimi dolami
3	Aladár	Baba	Kalifát Bagdad
4	Ferdinand	Mravec	Mravenisko
5	Ján	Polienko	Za siedmimi horami a siedmimi dolami
6	Juraj	Truľo	Za siedmimi horami a siedmimi dolami
7	Jana	Botková	NULL
8	Dana	Botková	NULL
9	Ján	Hlúpy	Za siedmimi horami a siedmimi dolami
10	Aladár	Miazga	NULL
11	Mikuláš	Myšiak	Hollywood
12	Donald	Káčer	Hollywood
13	Jozef	Námorník	NULL
14	Peter	Pan	Neverland

$\bowtie$

krajina	hlava_krajiny
Reťazec	Reťazec
1	1
Za siedmimi horami a siedmimi dolami	Drozdia Brada
Kalifát Bagdad	Harún al-Rašíd
Mravenisko	Z
Hollywood	Simba
Neverland	NULL
NULL	NULL

Posledným restom je tabuľka:

odbor	zameranie	gestor
Reťazec	Reťazec	Reťazec
0	0	0
metamatematika	metafyzika	Premúdra
metamatematika	matika	Miazga
informatematika	matika	Vševed
informatematika	metainformatika	Vševed

ktorej schéma

$$\text{schéma}(\{s_{\texttt{odbor}}, s_{\texttt{zameranie}}, s_{\texttt{gestor}}\}, \{\langle\{s_{\texttt{odbor}}, s_{\texttt{zameranie}}\}, \{s_{\texttt{gestor}}\}\rangle, \langle\{s_{\texttt{gestor}}\}, \{s_{\texttt{odbor}}\}\rangle\})$$

bola v 3NF, ale nie v BCNF. Podľa predchádzajúcej vety pre $X = \{s_{\texttt{gestor}}\}$, $Y^1 = \{s_{\texttt{odbor}}\}$ a $Y^2 = \{s_{\texttt{zameranie}}\}$ dostávame bezstratovú dekompozíciu na schémy

$$\text{schéma}(\{s_{\texttt{zameranie}}, s_{\texttt{gestor}}\}, \emptyset)$$

a

$$\text{schéma}(\{s_{\texttt{gestor}}, s_{\texttt{odbor}}\}, \{\langle\{s_{\texttt{gestor}}\}, \{s_{\texttt{odbor}}\}\rangle\}).$$

Pre našu tabuľku to teda bude dekompozícia:

zameranie	gestor
Reťazec	Reťazec
0	0
metafyzika	Premúdra
matika	Miazga
matika	Vševed
metainformatika	Vševed

$\bowtie$

gestor	odbor
Reťazec	Reťazec
0	0
Premúdra	metamatematika
Miazga	metamatematika
Vševed	informatematika

Bezstratová dekompozícia sa teda vie dôstojne vysporiadať so všetkými uvedenými duplicitami. Je to preto ona, čo **tvorí jadro databázovej teórie o správnom návrhu databázy**, a nie preceňované normálne formy, ktorým vždy nejaká duplicita unikne. Aj napriek tomu, že práve ony patria k imidžu databázistu. Asi ako bradavica k ježibabe...

4.4.Ú Úlohy

1 Dokážte, že prirozené spojenie je komutatívne a asociatívne.

2 Dokážte všetky lemy. Alebo aspoň všetky lemy v tejto podkapitole.

A
Apendix

A.1 Riešenia niektorých úloh

1.2-1

Problém je v tom, že priezviská majú rôznu dĺžku, treba preto pred každé z nich vsunúť príslušný počet nejakých neutrálnych znakov (medzier alebo (kvôli lepšej viditeľnosti) bodiek) tak, aby celková šírka bola 15 (ako je to deklarované v definícii stĺpca priezvisko). Pred priezvisko Hraško treba 9 bodiek, pred Baba 11 a pred Námorník 7, vo všeobecnosti teda bude vyhovovať $15 - \text{LENGTH}(\text{priezvisko})$ bodiek. Každý takýto bodkový reťazec je podreťazcom reťazca pozostávajúceho z 15 bodiek, výsledný dopyt teda bude:

```
SELECT SUBSTR('...............',1,15-LENGTH(priezvisko)) || priezvisko AS priezvisko
FROM student
```

Odpoveď:

PRIEZVISKO
.........Hraško
.........Šípová
...........Baba
.........Mravec
.......Polienko
..........Truľo
........Botková
........Botková
..........Hlúpy
.........Miazga
.........Myšiak
..........Káčer
.......Námorník

1.6-1

Za predpokladu, že tabuľka student vyzerá takto:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ružena	Šípová	žena	1984-02-01	1	1,22
Aladár	Baba	muž	1980-01-22	2	2,03
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Ján	Polienko	muž	1982-04-14	5	2,28
Juraj	Truľo	muž	1979-07-16	1	3,00
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40
Ján	Hlúpy	muž	NULL	2	3,00
Aladár	Miazga	muž	1987-12-22	3	2,06
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Jozef	Námorník	muž	1981-09-23	2	2,90
Peter	Pan	muž	2001-01-13	1	NULL
Grumpy	Dwarf	muž	1157-01-20	5	2,23
Doc	Dwarf	muž	1254-05-11	4	1,00
Bashful	Dwarf	muž	1197-06-12	4	2,02
Sleepy	Dwarf	muž	1292-12-02	1	4,50
Sneezy	Dwarf	muž	1251-07-13	2	4,28
Happy	Dwarf	muž	1173-02-11	3	1,00
Dopey	Dwarf	muž	1390-10-15	4	1,50
Sneh	Ulienka	žena	1967-01-01	1	1,00

stačí v modifikačnom príkaze použiť príkaz CASE (čo znamená, že funkcie tohto typu majú širšie uplatnenie než len v dopytoch):

```
UPDATE študent
SET meno =
      CASE meno
        WHEN 'Grumpy' THEN 'Dudroš'
        WHEN 'Doc' THEN 'Vedko'
        WHEN 'Bashful' THEN 'Plaško'
        WHEN 'Sleepy' THEN 'Spachtoš'
        WHEN 'Sneezy' THEN 'Hapčí'
        WHEN 'Happy' THEN 'Šťastko'
        WHEN 'Dopey' THEN 'Kýblik'
      END,
    priezvisko = 'Trpaslík'
WHERE priezvisko = 'Dwarf'
```

A naozaj, po tomto príkaze tabuľka vyzerá tak, ako kráľ chcel:

```
SELECT *
FROM študent
```

Odpoveď:

MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA	ROČNÍK	PRIEMER
Ján	Hraško	muž	1987-07-12	1	1,83
Ružena	Šípová	žena	1984-02-01	1	1,22
Aladár	Baba	muž	1980-01-22	2	2,03
Ferdinand	Mravec	muž	1984-03-03	3	1,00
Ján	Polienko	muž	1982-04-14	5	2,28
Juraj	Truľo	muž	1979-07-16	1	3,00
Jana	Botková	žena	1977-09-21	4	1,50
Dana	Botková	žena	1977-09-21	4	1,40
Ján	Hlúpy	muž	NULL	2	3,00
Aladár	Miazga	muž	1987-12-22	3	2,06
Mikuláš	Myšiak	muž	1983-06-06	5	1,66
Donald	Káčer	muž	1982-10-07	5	1,83
Jozef	Námorník	muž	1981-09-23	2	2,90
Peter	Pan	muž	2001-01-13	1	NULL
Dudroš	Trpaslík	muž	1157-01-20	5	2,23
Vedko	Trpaslík	muž	1254-05-11	4	1,00
Plaško	Trpaslík	muž	1197-06-12	4	2,02
Spachtoš	Trpaslík	muž	1292-12-02	1	4,50
Hapčí	Trpaslík	muž	1251-07-13	2	4,28
Šťastko	Trpaslík	muž	1173-02-11	3	1,00
Kýblik	Trpaslík	muž	1390-10-15	4	1,50
Sneh	Ulienka	žena	1967-01-01	1	1,00

2.5-1

Vypísať každého učiteľa raz by už nemalo robiť žiaden problém – jednoducho tabuľku `osoba` spojíme s tabuľkou `učiteľ`:

```
SELECT
  o.meno,
  o.priezvisko
FROM
  osoba AS o
    JOIN učiteľ AS u ON u.id = o.id
```

Odpoveď:

MENO	PRIEZVISKO
Gejza	Miazga
Matej	Múdry
Vasilisa	Premúdra
Hedviga	Baba
d'Eduard	Vševed

Keďže úloha je v kapitole o množinových operáciách, núka sa použitie zjednotenia. Zjednotíme teda predchádzajúci dopyt päťkrát, pričom do každej časti zjednotenia pridáme jeden konštantný stĺpec, ale vždy s inou hodnotou. Výsledok, pravdaže, usporiadame:

```
SELECT
  o.meno,
  o.priezvisko,
  1 AS exemplár
FROM
  osoba AS o
    JOIN učiteľ AS u ON u.id = o.id

UNION ALL

SELECT
  o.meno,
  o.priezvisko,
  2 AS exemplár
FROM
  osoba AS o
    JOIN učiteľ AS u ON u.id = o.id

UNION ALL

SELECT
  o.meno,
  o.priezvisko,
  3 AS exemplár
FROM
  osoba AS o
    JOIN učiteľ AS u ON u.id = o.id

UNION ALL

SELECT
  o.meno,
  o.priezvisko,
  4 AS exemplár
FROM
  osoba AS o
    JOIN učiteľ AS u ON u.id = o.id

UNION ALL

SELECT
  o.meno,
  o.priezvisko,
  5 AS exemplár
FROM
  osoba AS o
    JOIN učiteľ AS u ON u.id = o.id

ORDER BY 2, 1, 3
```

Odpoveď:

MENO	PRIEZVISKO	EXEMPLÁR
Gejza	Miazga	1
Gejza	Miazga	2
Gejza	Miazga	3
Gejza	Miazga	4
Gejza	Miazga	5
Matej	Múdry	1
Matej	Múdry	2
Matej	Múdry	3
Matej	Múdry	4
Matej	Múdry	5
Vasilisa	Premúdra	1
Vasilisa	Premúdra	2
Vasilisa	Premúdra	3
Vasilisa	Premúdra	4
Vasilisa	Premúdra	5
Hedviga	Baba	1
Hedviga	Baba	2
Hedviga	Baba	3
Hedviga	Baba	4
Hedviga	Baba	5
d'Eduard	Vševed	1
d'Eduard	Vševed	2
d'Eduard	Vševed	3
d'Eduard	Vševed	4
d'Eduard	Vševed	5

Inak škoda, že máme zakázané použiť `VALUES`. Stačilo by totiž vyrobiť si pomocou tohto príkazu tabuľku obsahujúcu hodnoty `1` až `5` a karteziánsky ju vynásobiť príslušnou časťou tabuľky `učiteľ`:

```
WITH poradové_číslo (číslo) as
(
  VALUES 1, 2, 3, 4, 5
)
SELECT
  o.meno,
  o.priezvisko,
  p.číslo AS exemplár
FROM
  osoba AS o
    JOIN učiteľ AS u ON u.id = o.id,
  poradové_číslo AS p
ORDER BY 2, 1, 3
```

2.5-2

Pripomeňme, že tabuľka `krajina` vyzerá takto:

ID	NÁZOV	HLAVA
1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
2	Kalifát Bagdad	Harún al-Rašíd
3	Mravenisko	Z
4	Hollywood	Simba
5	Neverland	NULL
6	Haliganda	NULL

a antikvárna tabuľka `štát` (predpokladanej rovnakej štruktúry) nech je:

ID	NÁZOV	HLAVA
1	Za siedmimi horami a siedmimi dolami	Matej Korvín
2	Kalifát Bagdad	Harún al-Rašíd
3	Mravenisko	Z
4	Hollywood	Simba
6	Kde bolo, tam bolo	Lomidrevo
7	Kráľovstvo krivých zrkadiel	Jágapap
8	Haliganda	NULL

Záznamy z oboch tabuliek sú troch typov:

1 nachádzajúce sa v oboch tabuľkách,
2 nachádzajúce sa len v tabuľke krajina,
3 nachádzajúce sa len v tabuľke štát.

Uvedomme si, že prvá skupina je vlastne prienik našich dvoch tabuliek, t. j.:

```
SELECT *
FROM krajina

INTERSECT

SELECT *
FROM štát

ORDER BY 1
```

Odpoveď:

ID	NÁZOV	HLAVA
2	Kalifát Bagdad	Harún al-Rašíd
3	Mravenisko	Z
4	Hollywood	Simba

Druhý typ zodpovedá rozdielu:

```
SELECT *
FROM krajina

EXCEPT

SELECT *
FROM štát

ORDER BY 1
```

s odpoveďou:

ID	NÁZOV	HLAVA
1	Za siedmimi horami a siedmimi dolami	Drozdia Brada
5	Neverland	NULL
6	Haliganda	NULL

a tretí opačnému rozdielu:

```
SELECT *
FROM štát

EXCEPT

SELECT *
FROM krajina

ORDER BY 1
```

s odpoveďou:

ID	NÁZOV	HLAVA
1	Za siedmimi horami a siedmimi dolami	Matej Korvín
6	Kde bolo, tam bolo	Lomidrevo
7	Kráľovstvo krivých zrkadiel	Jágapap
8	Haliganda	NULL

Prvá skupina je absolútne bezproblémová, tam sú obe tabuľky v totálnej zhode. Ako je to však so záznamami zvyšných dvoch typov? Keďže je tabuľka krajina pre nás prioritná a údaje z nej sú pre nás dôveryhodnejšie, všetky diskrepancie budeme riešiť v jej prospech (hoci možno s menšou istotou než predtým). Okrem iného to znamená to, že všetky záznamy druhého typu ponecháme v tabuľke krajina bez zmeny. Ostávajú záznamy tretieho typu, ktoré opäť rozdelíme na disjunktné skupinky (tu pripomeňme, že podľa definície sú ako id, tak názov jednoznačné):

1 V tabuľke krajina neexistuje ani záznam krajiny s rovnakým identifikátorom, ani záznam krajiny s rovnakým názvom:

```
SELECT *
FROM štát AS š
WHERE
      NOT id IN (SELECT id FROM krajina)
  AND NOT názov IN (SELECT názov FROM krajina)
```

Odpoveď:

ID	NÁZOV	HLAVA
7	Kráľovstvo krivých zrkadiel	Jágapap

2 V tabuľke krajina existuje záznam krajiny s rovnakým identifikátorom i názvom:

```
SELECT š.*
FROM
  štát AS š,
  krajina AS k
WHERE
      š.id = k.id
  AND š.názov = k.názov
```

Odpoveď:

ID	NÁZOV	HLAVA
1	Za siedmimi horami a siedmimi dolami	Matej Korvín

3 V tabuľke krajina existuje záznam krajiny s rovnakým názvom, ale rôznym identifikátorom:

```
SELECT š.*
FROM
  štát AS š,
  krajina AS k
WHERE
      š.id != k.id
  AND š.názov = k.názov
```

Odpoveď:

ID	NÁZOV	HLAVA
8	Haliganda	NULL

4 V tabuľke krajina existuje záznam krajiny s rovnakým identifikátorom, ale rôznym názvom:

```
SELECT š.*
FROM
  štát AS š,
  krajina AS k
WHERE
      š.id = k.id
  AND š.názov != k.názov
```

Odpoveď:

ID	NÁZOV	HLAVA
6	Kde bolo, tam bolo	Lomidrevo

Prvá skupinka je nekonfliktná, preto ju môžeme pokojne vložiť do tabuľky krajina.

Pri druhej môže byť rozpor len s nekľúčovým stĺpcom, ktorý je tu jediný – hlava. Povedali sme však, že tabuľke krajina veríme viac, preto jeho hodnotu meniť nebudeme.

Krajiny z tretej skupinky do tabuľky krajina vkladať nebudeme, lebo už tam sú (i keď s iným id). Treba však overiť, či sa na tento duplicitný identifikátor (v prípade Haligandy je to 8) neodkazujú záznamy v nejakých iných tabuľkách, a ak áno, zmeniť ho na správnu hodnotu z tabuľky krajina (pri Haligande na 6).

Prv než do tabuľky krajina skopírujeme údaje zo štvrtej skupinky, musíme zmeniť ich identifikátory na také, ktoré tam ešte nie sú (v prípade krajiny Kde bolo, tam bolo trebárs na 10). Nesmieme však popritom zabudnúť urobiť túto zmenu aj v prípadných záznamoch z iných tabuliek, ktoré sa na túto starú hodnotu odvolávali.

Aby bolo úplne po všetkom, treba ešte vykonať jednu protištátnu akciu: Bez obáv z obvinení z anarchizmu treba štát zrušiť (slovami piesne Spirituál Kvintetu „Musíš za svou pravdou, stát!“) a všetky cudzie kľúče, ktoré naň ukazovali, presmerovať na tabuľku krajina.

2.6-1

Kľúčové údaje budú zrejme v tabuľke zapísané. Ak si študent s identifikátorom x zapíše predmet s identifikátorom y, práve v tejto tabuľke sa objaví nový záznam, ktorého hodnota id_študent bude x a hodnota id_predmet y. Pri opakovaní dvojice (x, y) tam teda budú takéto záznamy aspoň dvakrát. Takže napíšme dopyt, ktorým nájdeme všetky takéto aspoň dvakrát sa opakujúce dvojice:

```
SELECT
  o.id,
  o.meno,
  o.priezvisko,
  p.názov AS predmet,
  COUNT(*) AS počet_prihlásení
FROM
  zapísané AS z
    JOIN osoba AS o ON z.id_študent = o.id
    JOIN predmet AS p ON z.id_predmet = p.id
GROUP BY
  o.id,
  o.meno,
  o.priezvisko,
  p.názov
HAVING COUNT(*) >= 2
```

Odpoveď:

ID	MENO	PRIEZVISKO	PREDMET	POČET_PRIHLÁSENÍ
107	Jana	Botková	Hlbokonepružný rozptyl leptónov na hadrónoch	2
109	Ján	Hlúpy	Zložité systémy	2
113	Jozef	Námorník	Zložité systémy	2

Teraz sa už iba stačí zbaviť posledného stĺpca:

```
WITH opakovaný_predmet AS
(
  SELECT
    o.id,
    o.meno,
    o.priezvisko,
    p.názov AS predmet,
    COUNT(*) AS počet_prihlásení
  FROM
    zapísané AS z
      JOIN osoba AS o ON z.id_študent = o.id
      JOIN predmet AS p ON z.id_predmet = p.id
  GROUP BY
```

```
    o.id,
    o.meno,
    o.priezvisko,
    p.názov
  HAVING COUNT(*) >= 2
)
SELECT
  id,
  meno,
  priezvisko,
  predmet
FROM opakovaný_predmet
```

Odpoveď:

ID	MENO	PRIEZVISKO	PREDMET
107	Jana	Botková	Hlbokonepružný rozptyl leptónov na hadrónoch
109	Ján	Hlúpy	Zložité systémy
113	Jozef	Námorník	Zložité systémy

Vidno, že Zložité systémy sú naozaj zložité...

2.6-2

Predpokladajme najprv zásadné porušenie prerekvizít, a to také, že podmieňujúci predmet nebol študentom ani len zapísaný (t. j. v súvislosti s týmto študentom sa v tabuľke `zapísané` vôbec nenachádza). Dopyt má potom tvar:

```
SELECT
  o.id,
  o.meno,
  o.priezvisko,
  p1.názov AS predmet1,
  p2.názov AS predmet2
FROM
  prerekvizita AS r
    JOIN predmet AS p1 ON r.id_podmieňujúci = p1.id
    JOIN predmet AS p2 ON r.id_podmieňovaný = p2.id
    JOIN zapísané AS z2 ON z2.id_predmet = p2.id
    JOIN osoba AS o ON z2.id_študent = o.id
WHERE NOT EXISTS
      (
        SELECT *
        FROM zapísané AS z1
        WHERE
              z1.id_predmet = p1.id
          AND z1.id_študent = o.id
      )
```

Odpoveď:

ID	MENO	PRIEZVISKO	PREDMET1	PREDMET2
107	Jana	Botková	Romológia	Teória hromadnej obsluhy
107	Jana	Botková	Fyzika DNA	Fázové prechody a kritické javy
107	Jana	Botková	Fyzika DNA	Nebeská mechanika
108	Dana	Botková	Fyzika DNA	Fázové prechody a kritické javy
108	Dana	Botková	Fyzika DNA	Nebeská mechanika
110	Aladár	Miazga	Nebeská mechanika	Teória hromadnej obsluhy
110	Aladár	Miazga	Romológia	Teória hromadnej obsluhy
113	Jozef	Námorník	Fázové prechody a kritické javy	Molekulový modeling

Teraz nás bude zaujímať druhá situácia – keď si študent zapísal aj podmieňujúci aj podmieňovaný predmet, ale ten prvý ešte neabsolvoval, alebo ho absolvoval až po úspešnom ukončení podmieňovaného predmetu. V dopyte si navyše všimnime aj dátumy (prípadných) ukončení a hodnotenia:

```
SELECT
  o.id,
  o.meno,
  o.priezvisko,
  p1.názov AS predmet1,
  z1.dátum_ukončenia AS dátum1,
  z1.hodnotenie AS h1,
  p2.názov AS predmet2,
  z2.dátum_ukončenia AS dátum2,
  z2.hodnotenie AS h2
FROM
  prerekvizita AS r
    JOIN predmet AS p1 ON r.id_podmieňujúci = p1.id
      JOIN zapísané AS z1 ON z1.id_predmet = p1.id
    JOIN predmet AS p2 ON r.id_podmieňovaný = p2.id
      JOIN zapísané AS z2 ON z2.id_predmet = p2.id AND z2.hodnotenie IS NOT NULL
    JOIN osoba AS o ON z1.id_študent = o.id AND z2.id_študent = o.id
WHERE
      z1.hodnotenie IS NULL
  OR  z1.dátum_ukončenia > z2.dátum_ukončenia
```

Odpoveď:

ID	MENO	PRIEZVISKO	PREDMET1	DÁTUM1	H1	PREDMET2	DÁTUM2	H2
104	Ferdinand	Mravec	Fázové prechody a kritické javy	2004-08-10	2	Molekulový modeling	2003-05-12	3
104	Ferdinand	Mravec	Fázové prechody a kritické javy	2004-08-10	2	Romológia	2004-05-10	1
105	Ján	Polienko	Romológia	NULL	NULL	Teória hromadnej obsluhy	2003-05-22	2
105	Ján	Polienko	Fázové prechody a kritické javy	2003-06-13	1	Molekulový modeling	2002-06-13	1
107	Jana	Botková	Fázové prechody a kritické javy	NULL	NULL	Molekulový modeling	2003-05-24	3
108	Dana	Botková	Fázové prechody a kritické javy	NULL	NULL	Molekulový modeling	2003-07-05	2
109	Ján	Hlúpy	Databázové systémy	2004-05-24	2	Fyzika DNA	2004-05-10	2
110	Aladár	Miazga	Fázové prechody a kritické javy	2004-06-05	3	Molekulový modeling	2003-05-16	2
110	Aladár	Miazga	Databázové systémy	NULL	NULL	Fyzika DNA	2003-06-10	2
111	Mikuláš	Myšiak	Romológia	NULL	NULL	Teória hromadnej obsluhy	2003-08-23	2
111	Mikuláš	Myšiak	Fázové prechody a kritické javy	2003-05-10	1	Molekulový modeling	2002-05-24	2
112	Donald	Káčer	Romológia	NULL	NULL	Teória hromadnej obsluhy	2003-10-24	2

Oba prípady porušenia prerekvizít tak môžeme vyjadriť zjednotením predchádzajúcich dopytov. Výber stĺpcov v prvom prípade však musíme doplniť, aby sa zhodoval s druhým, najlepšie by bolo doplniť stĺpce prázdnymi hodnotami. Žiaľ, napísať priamo do časti `SELECT` slovo `NULL` je, ako sme už spomínali, neprípustné, pretože nemá určený dátový typ. A práve tu je hviezdna chvíľa funkcie `NULLIF`! Použijeme hneď dve jej verzie, pre typ `INT` vezmeme trebárs `NULLIF(1,1)`, pre typ `DATE` hoci `NULLIF(CURRENT DATE,CURRENT DATE)`:

```
SELECT
  o.id,
  o.meno,
  o.priezvisko,
  p1.názov AS predmet1,
  NULLIF(CURRENT DATE,CURRENT DATE) AS dátum1,
  NULLIF(1,1) AS h1,
  p2.názov AS predmet2,
  NULLIF(CURRENT DATE,CURRENT DATE) AS dátum2,
  NULLIF(1,1) AS h2
FROM
  prerekvizita AS r
    JOIN predmet AS p1 ON r.id_podmieňujúci = p1.id
    JOIN predmet AS p2 ON r.id_podmieňovaný = p2.id
      JOIN zapísané AS z2 ON z2.id_predmet = p2.id
        JOIN osoba AS o ON z2.id_študent = o.id
WHERE NOT EXISTS
      (
        SELECT *
        FROM zapísané AS z1
        WHERE
              z1.id_predmet = p1.id
          AND z1.id_študent = o.id
      )

UNION ALL
```

```
SELECT
  o.id,
  o.meno,
  o.priezvisko,
  p1.názov AS predmet1,
  z1.dátum_ukončenia AS dátum1,
  z1.hodnotenie AS h1,
  p2.názov AS predmet2,
  z2.dátum_ukončenia AS dátum2,
  z2.hodnotenie AS h2
FROM
  prerekvizita AS r
    JOIN predmet AS p1 ON r.id_podmieňujúci = p1.id
      JOIN zapísané AS z1 ON z1.id_predmet = p1.id
    JOIN predmet AS p2 ON r.id_podmieňovaný = p2.id
      JOIN zapísané AS z2 ON z2.id_predmet = p2.id AND z2.hodnotenie IS NOT NULL
    JOIN osoba AS o ON z1.id_študent = o.id AND z2.id_študent = o.id
WHERE
      z1.hodnotenie IS NULL
  OR  z1.dátum_ukončenia > z2.dátum_ukončenia
```

Odpoveď:

ID	MENO	PRIEZVISKO	PREDMET1	DÁTUM1	H1	PREDMET2	DÁTUM2	H2
107	Jana	Botková	Romológia	NULL	NULL	Teória hromadnej obsluhy	NULL	NULL
107	Jana	Botková	Fyzika DNA	NULL	NULL	Fázové prechody a kritické javy	NULL	NULL
107	Jana	Botková	Fyzika DNA	NULL	NULL	Nebeská mechanika	NULL	NULL
108	Dana	Botková	Fyzika DNA	NULL	NULL	Fázové prechody a kritické javy	NULL	NULL
108	Dana	Botková	Fyzika DNA	NULL	NULL	Nebeská mechanika	NULL	NULL
110	Aladár	Miazga	Nebeská mechanika	NULL	NULL	Teória hromadnej obsluhy	NULL	NULL
110	Aladár	Miazga	Romológia	NULL	NULL	Teória hromadnej obsluhy	NULL	NULL
113	Jozef	Námorník	Fázové prechody a kritické javy	NULL	NULL	Molekulový modeling	NULL	NULL
104	Ferdinand	Mravec	Fázové prechody a kritické javy	2004-08-10	2	Molekulový modeling	2003-05-12	3
104	Ferdinand	Mravec	Fázové prechody a kritické javy	2004-08-10	2	Romológia	2004-05-10	1
105	Ján	Polienko	Romológia	NULL	NULL	Teória hromadnej obsluhy	2003-05-22	2
105	Ján	Polienko	Fázové prechody a kritické javy	2003-06-13	1	Molekulový modeling	2002-06-13	1
107	Jana	Botková	Fázové prechody a kritické javy	NULL	NULL	Molekulový modeling	2003-05-24	3
108	Dana	Botková	Fázové prechody a kritické javy	NULL	NULL	Molekulový modeling	2003-07-05	2
109	Ján	Hlúpy	Databázové systémy	2004-05-24	2	Fyzika DNA	2004-05-10	2
110	Aladár	Miazga	Fázové prechody a kritické javy	2004-06-05	3	Molekulový modeling	2003-05-16	2
110	Aladár	Miazga	Databázové systémy	NULL	NULL	Fyzika DNA	2003-06-10	2
111	Mikuláš	Myšiak	Romológia	NULL	NULL	Teória hromadnej obsluhy	2003-08-23	2
111	Mikuláš	Myšiak	Fázové prechody a kritické javy	2003-05-10	1	Molekulový modeling	2002-05-24	2
112	Donald	Káčer	Romológia	NULL	NULL	Teória hromadnej obsluhy	2003-10-24	2

2.6-3

Súc poučení príkladom o dvojiciach napojíme na seba tri exempláre tej istej tabuľky, pričom nezabúdame na to, že ľudia v trojici majú byť usporiadaní:

```
SELECT
  o1.id AS id1,
  o1.meno AS meno1,
  o1.priezvisko AS priezvisko1,
  o2.id AS id2,
  o2.meno AS meno2,
  o2.priezvisko AS priezvisko2,
  o3.id AS id3,
  o3.meno AS meno3,
  o3.priezvisko AS priezvisko3,
  i.číslo AS izba
FROM
  izba AS i
    JOIN študent AS š1 ON š1.id_izba = i.id
      JOIN osoba AS o1 ON o1.id = š1.id
    JOIN študent AS š2 ON š2.id_izba = i.id
      JOIN osoba AS o2 ON o2.id = š2.id
    JOIN študent AS š3 ON š3.id_izba = i.id
      JOIN osoba AS o3 ON o3.id = š3.id
```

```
WHERE
      (
          (o1.priezvisko < o2.priezvisko)
        OR (o1.priezvisko = o2.priezvisko AND o1.meno < o2.meno)
        OR (o1.priezvisko = o2.priezvisko AND o1.meno = o2.meno AND o1.id < o2.id)
      )
  AND (
          (o2.priezvisko < o3.priezvisko)
        OR (o2.priezvisko = o3.priezvisko AND o2.meno < o3.meno)
        OR (o2.priezvisko = o3.priezvisko AND o2.meno = o3.meno AND o2.id < o3.id)
      )
ORDER BY 3, 2, 1, 6, 5, 4, 9, 8, 7
```

Odpoveď:

ID1	MENO1	PRIEZVISKO1	ID2	MENO2	PRIEZVISKO2	ID3	MENO3	PRIEZVISKO3	IZBA
108	Dana	Botková	102	Jana	Botková	102	Ružena	Šípová	101A
104	Ferdinand	Mravec	105	Ján	Polienko	106	Juraj	Truľo	242B

2.6-4

Stačí málo – vrátiť sa jednoducho k dvojiciam (bez opakovania) a položiť dodatočnú podmienku na rôznosť pohlavia:

```
SELECT
  o1.id AS id1,
  o1.meno AS meno1,
  o1.priezvisko AS priezvisko1,
  o2.id AS id2,
  o2.meno AS meno2,
  o2.priezvisko AS priezvisko2,
  i.číslo AS izba
FROM
  izba AS i
    JOIN študent AS š1 ON š1.id_izba = i.id
      JOIN osoba AS o1 ON o1.id = š1.id
    JOIN študent AS š2 ON š2.id_izba = i.id
      JOIN osoba AS o2 ON o2.id = š2.id
WHERE
      (
          (o1.priezvisko < o2.priezvisko)
        OR (o1.priezvisko = o2.priezvisko AND o1.meno < o2.meno)
        OR (o1.priezvisko = o2.priezvisko AND o1.meno = o2.meno AND o1.id < o2.id)
      )
  AND o1.pohlavie <> o2.pohlavie
ORDER BY 3, 2, 1, 6, 5, 4
```

Odpoveď:

ID1	MENO1	PRIEZVISKO1	ID2	MENO2	PRIEZVISKO2	IZBA

Všetko je teda v poriadku. Teraz už hej, ale keď bol omylom Aladár Baba vzhľadom na svoje priezvisko považovaný za dcéru Baby Jadvigy, t. j. keď bola situácia ako po vykonaní príkazu:

```
UPDATE osoba
SET pohlavie = 'žena'
WHERE id = 103
```

výsledok predchádzajúceho dopytu bol takýto:

ID1	MENO1	PRIEZVISKO1	ID2	MENO2	PRIEZVISKO2	IZBA
103	Aladár	Baba	103	Aladár	Miazga	354B

3.1-1

Vrátime sa k riešeniu úlohy 2.6-2 a jednoducho nahradíme tabuľku `prerekvizita` jej tranzitívnym uzáverom (a odstránime duplicity):

```
WITH nepriama_prerekvizita (id_podmieňovaný, id_podmieňujúci) AS
(
  SELECT *
  FROM prerekvizita

  UNION ALL

  SELECT
    p.id_podmieňovaný,
    n.id_podmieňujúci
  FROM
    prerekvizita AS p,
    nepriama_prerekvizita AS n
  WHERE p.id_podmieňujúci = n.id_podmieňovaný
)

SELECT DISTINCT
  o.id,
  o.meno,
  o.priezvisko,
  p1.názov AS podmieňujúci_predmet,
  NULLIF(CURRENT DATE,CURRENT DATE) AS dátum_ukončenia,
  NULLIF(1,1) AS hodnotenie,
  p2.názov AS podmieňovaný_predmet,
  NULLIF(CURRENT DATE,CURRENT DATE) AS dátum_ukončenia,
  NULLIF(1,1) AS hodnotenie
FROM
  nepriama_prerekvizita AS r
    JOIN predmet AS p1 ON r.id_podmieňujúci = p1.id
    JOIN predmet AS p2 ON r.id_podmieňovaný = p2.id
      JOIN zapísané AS z2 ON z2.id_predmet = p2.id
        JOIN osoba AS o ON z2.id_študent = o.id
WHERE NOT EXISTS
      (
        SELECT *
        FROM zapísané AS z1
        WHERE
              z1.id_predmet = p1.id
          AND z1.id_študent = o.id
      )

UNION

SELECT DISTINCT
  o.id,
  o.meno,
  o.priezvisko,
  p1.názov AS podmieňujúci_predmet,
  z1.dátum_ukončenia,
  z1.hodnotenie,
  p2.názov AS podmieňovaný_predmet,
  z2.dátum_ukončenia,
  z2.hodnotenie
FROM
  nepriama_prerekvizita AS r
    JOIN predmet AS p1 ON r.id_podmieňujúci = p1.id
      JOIN zapísané AS z1 ON z1.id_predmet = p2.id
    JOIN predmet AS p2 ON r.id_podmieňovaný = p2.id
      JOIN zapísané AS z2 ON z2.id_predmet = p2.id AND z2.hodnotenie IS NOT NULL
        JOIN osoba AS o ON z2.id_študent = o.id
WHERE
      z1.hodnotenie IS NULL
  OR  z1.dátum_ukončenia > z2.dátum_zápisu
```

Odpoveď radšej neuvádzame, má 357 riadov... No teda, toľko prehreškov?! Naozaj najvyšší čas začať používať informačný systém!

3.1-2

Táto úloha pripomína úlohu 2.5.1, tam však malo naše n konkrétnu hodnotu. Tam použité zjednocovanie je teda nepoužiteľné, počet `UNION ALL` by bol totiž $n-1$, a takúto bezbrehosť si nemôžeme dovoliť. Ani druhé jej (pomiňme, že zakázané) riešenie s tabuľkou obsahujúcou čísla `1` až `5` priamo použiť nemôžeme, veď počet explicitne uvedených hodnôt v tomto príkaze pomocou `VALUES` by musel byť n, čo je neznáme.

No dobre, a nemôžeme si takúto tabuľku obsahujúcu čísla do n vyrobiť inak? A keďže táto úloha asi nie je v kapitole o tranzitívnom uzávere náhodou, skúsme ho využiť.

V prvom kroku indukcie použijeme jednoducho príkaz `VALUES`, ktorý vráti `1`. V druhom indukčnom kroku potom budeme postupne zväčšovať toto číslo po jednotkách až do potrebnej hranice n. Takže dopyt produkujúci tabuľku obsahujúcu čísla `1` až trebárs po `5` vyzerá takto:

```
WITH poradové_číslo (číslo) as
(
  VALUES 1

  UNION ALL

  SELECT číslo + 1
  FROM poradové_číslo
  WHERE číslo < 5
)
SELECT *
FROM poradové_číslo
```

Odpoveď:

ČÍSLO
1
2
3
4
5

Ako sme už povedali, teraz už stačí touto tabuľkou vynásobiť zoznam učiteľov:

```
WITH poradové_číslo (číslo) as
(
  VALUES 1

  UNION ALL

  SELECT číslo + 1
  FROM poradové_číslo
  WHERE číslo < 5
)
SELECT
  o.meno,
  o.priezvisko,
  p.číslo AS exemplár
FROM
  osoba AS o
    JOIN učiteľ AS u ON u.id = o.id,
  poradové_číslo AS p
ORDER BY 2, 1, 3
```

Odpoveď:

MENO	PRIEZVISKO	EXEMPLÁR
Gejza	Miazga	1
Gejza	Miazga	2
Gejza	Miazga	3
Gejza	Miazga	4
Gejza	Miazga	5
Matej	Múdry	1
Matej	Múdry	2
Matej	Múdry	3
Matej	Múdry	4
Matej	Múdry	5
Vasilisa	Premúdra	1
Vasilisa	Premúdra	2
Vasilisa	Premúdra	3
Vasilisa	Premúdra	4
Vasilisa	Premúdra	5
Hedviga	Baba	1
Hedviga	Baba	2
Hedviga	Baba	3
Hedviga	Baba	4
Hedviga	Baba	5
d'Eduard	Vševed	1
d'Eduard	Vševed	2
d'Eduard	Vševed	3
d'Eduard	Vševed	4
d'Eduard	Vševed	5

3.1-3

Nech tabuľka `študent` po pridaní spomínaného stĺpca vyzerá takto:

ID	ID_SKUPINA	ROČNÍK	ID_BYDLSIKO	ID_IZBA	ID_ZDEDENÉ_OD
101	1	1	1	NULL	113
102	1	1	1	1	109
103	2	2	2	4	104
104	2	3	3	3	108
105	3	5	1	3	NULL
106	3	1	1	3	103
107	3	4	NULL	1	105
108	3	4	NULL	1	111
109	2	2	1	2	112
110	2	3	NULL	4	107
111	1	5	4	5	NULL
112	1	5	4	5	NULL
113	1	2	NULL	NULL	110
114	2	1	5	NULL	NULL

Z tabuľky tak môžeme prečítať, že napríklad študent s číslom 113 zdedil knihy od študenta s číslom 110, ten (rok predtým) od študenta 107 a ten (ďalší rok dozadu) od 105. Prázdna hodnota v riadku študenta 105 zas znamená, že učebnice, ktoré dostal on, boli nové. To celé znamená, že ak by číslo daného študenta bolo 113, výsledná tabuľka by obsahovala práve 110, 107 a 105. Medzi študentmi 113 a 107 teda existuje väzba, ktorá nie je priamo uvedená v tabuľke `študent`, ale môžeme ju získať tranzitivitou cez 110. Podobne je to s väzbou 113 k 105, tu však máme dva tranzity – 110 a 107.

Už vieme, že takéto prechody vieme dostať rekurziou. V prvom indukčnom kroku vypíšeme väzby dané priamo v tabuľke, v druhom (ktorý sa bude iterovať) existujúce (priame či nepriame) väzby predĺžime. Navyše tak môžeme (v treťom stĺpci) evidovať počet odovzdávok kníh, kým sa dostali od druhého študenta k prvému. Výsledný dopyt teda bude takýto:

```
WITH nepriama_väzba (id, id_zdedené_od, počet_odovzdávok) AS
(
  SELECT
    id,
    id_zdedené_od,
    1
  FROM študent
  WHERE id_zdedené_od IS NOT NULL

  UNION ALL

  SELECT
    š.id,
    n.id_zdedené_od,
    n.počet_odovzdávok + 1
  FROM
    študent AS š,
    nepriama_väzba AS n
  WHERE š.id_zdedené_od = n.id
)
SELECT *
FROM nepriama_väzba
```

Odpoveď:

ID	ID_ZDEDENÉ_OD	POČET_ODOVZDÁVOK
101	113	1
102	109	1
103	104	1
104	108	1
106	103	1
107	105	1
108	111	1
109	112	1
110	107	1
113	110	1
106	104	2
103	108	2
110	105	2
104	111	2
102	112	2
113	107	2
101	110	2
106	108	3
113	105	3
103	111	3
101	107	3
101	105	4
106	111	4

Aby sme úlohu splnili do bodky, treba, pravdaže, dopyt doplniť podmienkou `WHERE id =` x, kde x je číslo daného študenta. Ak by sme namiesto čísel študentov chceli pracovať s ich menami, pomocnú tabuľku meniť nemusíme, stačí upraviť koncový dopyt napojením tých správnych tabuliek. Asi takto:

```
WITH nepriama_väzba (id, id_zdedené_od, počet_odovzdávok) AS
(
  SELECT
    id,
    id_zdedené_od,
    1
  FROM študent
  WHERE id_zdedené_od IS NOT NULL

  UNION ALL
```

```
  SELECT
    š.id,
    n.id_zdedené_od,
    n.počet_odovzdávok + 1
  FROM
    študent AS š,
    nepriama_väzba AS n
  WHERE š.id_zdedené_od = n.id
)
SELECT
  o1.id AS id1,
  o1.meno AS meno1,
  o1.priezvisko AS priezvisko1,
  n.počet_odovzdávok,
  o2.id id2,
  o2.meno meno2,
  o2.priezvisko AS priezvisko2
FROM
  nepriama_väzba AS n
    JOIN osoba AS o1 ON o1.id = n.id
    JOIN osoba AS o2 ON o2.id = n.id_zdedené_od
ORDER BY 1, 4
```

Odpoveď:

ID1	MENO1	PRIEZVISKO1	POČET_ODOVZDÁVOK	ID2	MENO2	PRIEZVISKO2
101	Ján	Hraško	1	113	Jozef	Námorník
101	Ján	Hraško	2	110	Aladár	Miazga
101	Ján	Hraško	3	107	Jana	Botková
101	Ján	Hraško	4	105	Ján	Polienko
102	Ružena	Šípová	1	109	Ján	Hlúpy
102	Ružena	Šípová	2	112	Donald	Káčer
103	Aladár	Baba	1	104	Ferdinand	Mravec
103	Aladár	Baba	2	108	Dana	Botková
103	Aladár	Baba	3	111	Mikuláš	Myšiak
104	Ferdinand	Mravec	1	108	Dana	Botková
104	Ferdinand	Mravec	2	111	Mikuláš	Myšiak
106	Juraj	Truľo	1	103	Aladár	Baba
106	Juraj	Truľo	2	104	Ferdinand	Mravec
106	Juraj	Truľo	3	108	Dana	Botková
106	Juraj	Truľo	4	111	Mikuláš	Myšiak
107	Jana	Botková	1	105	Ján	Polienko
108	Dana	Botková	1	111	Mikuláš	Myšiak
109	Ján	Hlúpy	1	112	Donald	Káčer
110	Aladár	Miazga	1	107	Jana	Botková
110	Aladár	Miazga	2	105	Ján	Polienko
113	Jozef	Námorník	1	110	Aladár	Miazga
113	Jozef	Námorník	2	107	Jana	Botková
113	Jozef	Námorník	3	105	Ján	Polienko

V tomto riešení sme študentov evidovali po jednom, teda ku každému študentovi sme dostali toľko riadkov, po koľkých kolegoch (priamo či nepriamo) knihy dedil. Okrem takéhoto „vertikálneho“ riešenia je možné aj riešenie „horizontálne“, keď ku každému študentovi vypíšeme celú postupnosť jeho „predkov“. V prvom kroku indukcie vybavíme študentov, ktorí dostali nové knihy, a teda ich postupnosť je prázdna (tu si všimnime, že musíme zmenou dátového typu druhého stĺpca pomocnú tabuľku pripraviť tak, aby bola schopná prijať aj dlhšie postupnosti než prázdnu). V druhom kroku potom využijeme fakt, že postupnosť študenta je postupnosťou toho, po kom knihy dedil, predĺžená o (niektoré) údaje tohto jeho predka:

```
WITH postupnosť (id, postupnosť) AS
(
  SELECT
    id,
    CAST ('' AS VARCHAR(100))
  FROM študent
  WHERE id_zdedené_od IS NULL

  UNION ALL

  SELECT
    dedič.id,
    p.postupnosť || ' / ' || predok.meno || ' ' || predok.priezvisko
  FROM
    študent AS dedič,
    postupnosť AS p,
    osoba AS predok
  WHERE
        dedič.id_zdedené_od = p.id
    AND predok.id = p.id
)
SELECT
  o.id,
  o.meno,
  o.priezvisko,
  p.postupnosť
FROM
  postupnosť AS p
    JOIN osoba AS o ON o.id = p.id
```

Odpoveď:

ID	MENO	PRIEZVISKO	POSTUPNOSŤ
101	Ján	Hraško	/ Ján Polienko / Jana Botková / Aladár Miazga / Jozef Námorník
102	Ružena	Šípová	/ Donald Káčer / Ján Hlúpy
103	Aladár	Baba	/ Mikuláš Myšiak / Dana Botková / Ferdinand Mravec
104	Ferdinand	Mravec	/ Mikuláš Myšiak / Dana Botková
105	Ján	Polienko	
106	Juraj	Truľo	/ Mikuláš Myšiak / Dana Botková / Ferdinand Mravec / Aladár Baba
107	Jana	Botková	/ Ján Polienko
108	Dana	Botková	/ Mikuláš Myšiak
109	Ján	Hlúpy	/ Donald Káčer
110	Aladár	Miazga	/ Ján Polienko / Jana Botková
111	Mikuláš	Myšiak	
112	Donald	Káčer	
113	Jozef	Námorník	/ Ján Polienko / Jana Botková / Aladár Miazga
114	Peter	Pan	

3.1-4

Keby v zadaní nefigurovalo slovo „pracovných“, stačilo by využiť navzájom inverzné funkcie DAYS a DATE, takže dopyt by trebárs pre dátum 24. 3. 2005 mohol vyzerať takto:

```
VALUES DATE(DAYS('24.3.2005') + 5)
```

Odpoveď:

2005-03-29

Toto však, žiaľ, nie je náš prípad – niektoré z týchto troch dní totiž môžu byť víkendové alebo sviatočné, a potom treba lehotu príslušne predĺžiť.

Predpokladajme, že tabuľka sviatok obsahuje takýto stĺpec (a nech sú vypísané práve všetky jeho hodnoty z roku 2005):

...	DEŇ	...
...	...	...
...	2005-01-01	...
...	2005-01-06	...
...	2005-03-25	...
...	2005-03-27	...
...	2005-03-28	...
...	2005-05-01	...
...	2005-05-08	...
...	2005-07-05	...
...	2005-08-29	...
...	2005-09-01	...
...	2005-09-15	...
...	2005-11-01	...
...	2005-11-17	...
...	2005-12-24	...
...	2005-12-25	...
...	2005-12-26	...
...	...	...

Opäť pomôže rekurzia. Vytvoríme si ňou totiž tabuľku, kde budeme evidovať charakteristiku každého dňa po zadanom dni až do okamihu, keď počet započítaných pracovných dní dosiahne číslo 5. V prvom indukčnom kroku vybavíme zadaný deň (nezabudneme pri tom konverziou určiť rozsah komentára tak, aby sa doň vošli i ďalšie). Keďže sa do našej lehoty nezaratúva, počet pracovných dní doteraz bude 0, a opatríme ho aj patričným komentárom. Druhý krok indukcie, v ktorom sa budeme opakovane zaoberať ďalším dňom, sa rozpadne na tri disjunktné časti – v prvej pôjde o sviatky z tabuľky sviatok (včítane víkendových), v druhom o víkendové, ale nesviatočné dni a v treťom o dni ostatné, t. j. pracovné. V prvých dvoch prípadoch sa počet doterajších pracovných dní doteraz nezmení, v treťom sa o jeden zväčší (práve sme totiž našli ďalší). Zároveň si zaznačíme aj príslušný komentár:

```
WITH hľadaný_deň (deň, komentár, počet_doteraz) AS
(
  VALUES
    (DATE('24.3.2005'), CAST('zadaný deň' AS VARCHAR(50)), 0)

  UNION ALL

  SELECT
    DATE(DAYS(hľadaný_deň.deň) + 1),
    'voľný - sviatok',
    počet_doteraz
  FROM
    hľadaný_deň,
    sviatok
  WHERE
        DATE(DAYS(hľadaný_deň.deň) + 1) = sviatok.deň
    AND počet_doteraz + 1 <= 5

  UNION ALL

  SELECT
    DATE(DAYS(deň) + 1),
    'voľný - víkend',
    počet_doteraz
  FROM hľadaný_deň
  WHERE
        NOT DATE(DAYS(deň) + 1) IN (SELECT deň FROM sviatok)
    AND DAYOFWEEK(DATE(DAYS(deň) + 1)) IN (7,1)
    AND počet_doteraz + 1 <= 5

  UNION ALL
```

```
  SELECT
    DATE(DAYS(deň) + 1),
    'pracovný',
    počet_doteraz + 1
  FROM hľadaný_deň
  WHERE
        NOT DATE(DAYS(deň) + 1) IN (SELECT deň FROM sviatok)
    AND DAYOFWEEK(DATE(DAYS(deň) + 1)) NOT IN (7,1)
    AND počet_doteraz + 1 <= 5
)
SELECT *
FROM hľadaný_deň
ORDER BY 1
```

DEŇ	KOMENTÁR	POČET_DOTERAZ
2005-03-24	zadaný deň	0
2005-03-25	voľný - sviatok	0
2005-03-26	voľný - víkend	0
2005-03-27	voľný - sviatok	0
2005-03-28	voľný - sviatok	0
2005-03-29	pracovný	1
2005-03-30	pracovný	2
2005-03-31	pracovný	3
2005-04-01	pracovný	4
2005-04-02	voľný - víkend	4
2005-04-03	voľný - víkend	4
2005-04-04	pracovný	5

Aby sme dostáli zadaniu, treba vypísať iba pracovný deň s poradovým číslom 5, treba, isteže, záverečný dopyt upraviť do podoby:

```
SELECT deň
FROM hľadaný_deň
WHERE
      komentár = 'pracovný'
  AND počet_doteraz = 5
```

3.3-1

Pri vložení údajov o študentovi do tabuliek `osoba` a následne `študent` by sa teda mal do tabuľky `zapísané` automaticky vložiť záznam hovoriaci, že tento študent si zapísal predmet s identifikátorom 12. Keďže musíme dbať na primárny kľúč, hodnota identifikátora nového záznamu bude presahovať všetky doterajšie hodnoty identifikátorov existujúcich záznamov. Trigger teda bude vyzerať takto:

```
CREATE TRIGGER zápis_dbs_automat
AFTER INSERT ON študent
REFERENCING NEW AS n
FOR EACH ROW
MODE DB2SQL
INSERT INTO zapísané
VALUES (VALUE((SELECT MAX(id) FROM zapísané), 0) + 1, n.id, 12, CURRENT DATE, NULL, NULL)
```

A naozaj, po vložení novej študentky:

```
INSERT INTO osoba
VALUES (115, 'Kristína', 'Miazgová', 'žena', '11.11.1982')
;
INSERT INTO študent
VALUES (115, 1, 1, NULL, 1)
```

bude relevantná časť tabuľky `zapísané` vyzerať takto:

```
SELECT *
FROM zapísané
WHERE id > 105
```

Odpoveď:

ID	ID_ŠTUDENT	ID_PREDMET	DÁTUM_ZÁPISU	DÁTUM_UKONČENIA	HODNOTENIE
106	115	12	dnešný dátum	NULL	NULL

Do tabuľky teda pribudol nový záznam, ktorého `id` je väčšie než doterajšie maximum 105, a teda rôzne od všetkých ostatných. Týka sa novej študentky (za `n.id` sa dosadil jej identifikátor 115 z tabuľky `študent`) a našich Databázových systémov (s identifikátorom 12).

3.3-2

Dvojnásobné neúspešné ukončenie znamená, že sa v tabuľke `zapísané` vyskytne dvakrát tá istá kombinácia dotyčného študenta a nejakého predmetu, pričom `hodnotenie` bude `NULL`, ale `dátum_ukončenia` nie. V takom prípade máme vyhodiť študenta z tabuľky `osoba` (vymazanie jeho údajov z ďalších relevantných tabuliek sa uskutoční automaticky). Kontrola bude prebiehať pri každom pokuse modifikovať `dátum_ukončenia` alebo `hodnotenie`. Trigger potom bude vyzerať takto:

```
CREATE TRIGGER vyhodenie_flákača
AFTER UPDATE OF hodnotenie, dátum_ukončenia ON zapísané
REFERENCING NEW AS n
FOR EACH ROW
MODE DB2SQL
WHEN
(
  (
    SELECT MAX(počet)
    FROM
    (
      SELECT
        id_predmet,
        id_študent,
        COUNT(id) AS počet
      FROM zapísané
      WHERE
            hodnotenie IS NULL
        AND dátum_ukončenia IS NOT NULL
      GROUP BY
        id_predmet,
        id_študent
    ) AS pom
    WHERE id_študent = n.id_študent
  )
  >= 2
)
DELETE FROM osoba
WHERE id = n.id_študent
```

Dopyt v podmienke `WHEN`, žiaľ, nemôže byť napísaný pomocou `WITH`. Ďalším obmedzením je, že vo vnútornom dopyte, ktorý počíta počet zápisov, už nemožno použiť odkaz `n` na nový záznam, preto tam rátame počty pre všetkých študentov a obmedzujúca podmienka je až vo vonkajšom dopyte. Považujme to za nedostatok DB2. Čo už...

Všimnime si, ako stojí študent Ján Hlúpy (s číslom 109) z predmetu Zložité systémy (s číslom 4):

```
SELECT *
FROM zapísané
WHERE
      id_študent = 109
  AND id_predmet = 4
ORDER BY dátum_zápisu
```

Odpoveď:

ID	ID_ŠTUDENT	ID_PREDMET	DÁTUM_ZÁPISU	DÁTUM_UKONČENIA	HODNOTENIE
20	109	4	2002-09-01	2003-08-10	NULL
21	109	4	2003-09-02	NULL	NULL

Vidíme, že zle – prvýkrát ho neurobil (lebo `hodnotenie` je prázdne, ale `dátum_ukončenia` nie), teraz ho opakuje. Povedzme, že mu to nevyšlo ani na druhýkrát, čo znamená modifikáciu stĺpca `hodnotenie` na prázdnu hodnotu a hlavne následné nastavenie hodnoty `dátum_ukončenia` na neprázdnu hodnotu:

```
UPDATE zapísané
SET
  hodnotenie = NULL,
  dátum_ukončenia = CURRENT DATE
WHERE id = 21
```

Tým sa však inicializuje náš trigger – a po Jánovi Hlúpom neostalo ani pamiatky:

```
SELECT *
FROM osoba
WHERE id = 109
```

Odpoveď:

ID	MENO	PRIEZVISKO	POHLAVIE	DÁTUM_NARODENIA

a potom samozrejme (vzhľadom na charakter príslušných cudzích kľúčov) aj:

```
SELECT *
FROM študent
WHERE id = 109
```

Odpoveď:

ID	ID_SKUPINA	ROČNÍK	ID_KRAJINA	ID_IZBA

a:

```
SELECT *
FROM zapísané
WHERE id_študent = 109
```

Odpoveď:

ID	ID_ŠTUDENT	ID_PREDMET	DÁTUM_ZÁPISU	DÁTUM_UKONČENIA	HODNOTENIE

Smola, mal sa viac učiť...

3.3-3

Či sa zmenila situácia, budeme kontrolovať po každej zmene stĺpca `hodnotenie` v tabuľke `zapísané`, a spúšťaná akcia sa bude týkať stĺpca `ročník` v tabuľke `študent`, ktorý sa zväčší o jeden. Takže:

```
CREATE TRIGGER zvýšenie_ročníka
AFTER UPDATE OF hodnotenie ON zapísané
REFERENCING NEW AS n
FOR EACH ROW
MODE DB2SQL
```

```
WHEN
(
  (
    SELECT SUM(p.kredit) AS počet_kreditov
    FROM
      zapísané AS z
        JOIN predmet AS p ON z.id_predmet = p.id
    WHERE
          z.id_študent = n.id_študent
      AND z.hodnotenie IS NOT NULL
  )
  >= 12 *
  (
    SELECT ročník
    FROM študent
    WHERE n.id_študent = id
  )
)
UPDATE študent
SET ročník = ročník + 1
WHERE id = n.id_študent
```

Pozrime sa na ročník a počet kreditov Aladára Miazgu (s číslom 110):

```
SELECT ročník
FROM študent
WHERE id = 110
;
SELECT SUM(p.kredit) AS počet_kreditov
FROM
  zapísané AS z
    JOIN predmet AS p ON z.id_predmet = p.id
WHERE
      z.dátum_ukončenia IS NULL
  AND z.id_študent = 110
```

Odpovede:

ROČNÍK
3

POČET_KREDITOV
33

Chýbajú mu teda 3 kredity na to, aby sa stal štvrtákom. Jeho neuzavreté predmety sú:

```
SELECT
  z.id AS id_zapísané,
  z.id_predmet,
  p.názov AS predmet,
  p. kredit
FROM
  zapísané AS z
    JOIN predmet AS p ON z.id_predmet = p.id
WHERE
      z.dátum_ukončenia IS NULL
  AND z.id_študent = 110
```

Odpoveď:

ID_ZAPÍSANÉ	ID_PREDMET	KREDIT
45	12	5
51	8	4

Ak teda úspešne uzavrie aspoň jeden z týchto predmetov, počet jeho kreditov dosiahne aspoň 36, čo je trojnásobok čísla jeho ročníka, a stane sa štvrtákom. Nech je to ten druhý a doprajme mu jednotku:

```
UPDATE zapísané
SET hodnotenie = 1
WHERE id = 51
```

A teraz sa pozrime na počet jeho kreditov:

```
SELECT SUM(p.kredit) AS počet_kreditov
FROM
  zapísané AS z
    JOIN predmet AS p ON z.id_predmet = p.id
WHERE
      z.dátum_ukončenia IS NULL
  AND z.id_študent = 110
```

Odpoveď:

POČET_KREDITOV
37

A na jeho ročník:

```
SELECT ročník
FROM študent
WHERE id = 110
```

Odpoveď:

ROČNÍK
4

Náš Aladár Miazga teda postúpil do vyššieho ročníka. Blahoželáme!

3.3-4

Uvedomme si, že stačí kontrolovať priame prerekvizity (tranzitívny uzáver prázdnej relácie je totiž prázdny). Podmienka vo WHEN hovorí, že počet všetkých prerekvizít predmetu je väčší než počet tých podmieňujúcich predmetov, ktoré študent úspešne absolvoval, a teda mu nutne nejaký chýba:

```
CREATE TRIGGER bacha_prerekvizity
NO CASCADE BEFORE INSERT ON zapísané
REFERENCING NEW AS n
FOR EACH ROW
MODE DB2SQL
WHEN
(
  (
    SELECT COUNT(*)
    FROM prerekvizita AS p
    WHERE p.id_podmieňovaný = n.id_predmet
  )
  >
  (
    SELECT COUNT(DISTINCT p.id_podmieňujúci)
    FROM
      prerekvizita AS p
        JOIN zapísané AS z ON p.id_podmieňujúci = z.id_predmet
    WHERE
          p.id_podmieňovaný = n.id_predmet
      AND z.id_študent = n.id_študent
      AND z.hodnotenie IS NOT NULL
  )
)
SIGNAL SQLSTATE 'KDEZE' ('Kdeže! Najprv treba absolvovať prerekvizity!')
```

Všimnime si, ako na tom stojí študent s číslom 101:

```
SELECT *
FROM zapísané
WHERE id_študent = 101
```

Odpoveď:

ID	ID_ŠTUDENT	ID_PREDMET	DÁTUM_ZÁPISU	DÁTUM_UKONČENIA	HODNOTENIE
1	101	1	2003-09-01	2004-05-20	2
2	101	4	2003-09-01	NULL	NULL
3	101	12	2003-09-01	NULL	NULL

Vzhľadom na to, že má síce zapísaný predmet s číslom 12, ale ešte ho neabsolvoval, nemôže si zapísať predmet s číslom 10, ktorého prerekvizita je práve 12. Takýto pokus:

```
INSERT INTO zapísané
VALUES (200, 101, 10, CURRENT DATE, NULL, NULL)
```

teda dopadne neúspešne a systém ho okrem iného takto vyhreší:

```
... Application raised error with diagnostic text:
"Kdeže! Najprv treba absolvovať prerekvizity!".
SQLSTATE=KDEZE
```

Podobne dopadne aj takýto pokus:

```
INSERT INTO zapísané
VALUES (200, 101, 9, CURRENT DATE, NULL, NULL)
```

Študent sa totiž pokúša zapísať si predmet s číslom 9, ktorého prerekvizitu s číslom 2 si ani len nezapísal.

3.3-5

Stačí sledovať situáciu po modifikácii stĺpca `id_izba` tabuľky `študent` a zisťovať, či počet študentov v dotyčnej izbe (včítane práve presťahovaného) nepresiahol jej kapacitu. Výsledný trigger má teda podobu:

```
CREATE TRIGGER husto
AFTER UPDATE OF id_izba ON študent
REFERENCING NEW AS n
FOR EACH ROW
MODE DB2SQL
WHEN
(
  (
    SELECT COUNT(*)
    FROM študent AS š
    WHERE š.id_izba = n.id_izba
  )
  >
  (
    SELECT d.kapacita
    FROM
      druh_izby AS d
        JOIN izba AS i ON i.id_druh = d.id
    WHERE i.id = n.id_izba
  )
)
SIGNAL SQLSTATE 'HUSTO' ('Maj rozum! To má spať na zemi?')
```

A naozaj, pri pokuse presťahovať študenta s číslom 104 do plne obsadenej izby s číslom 4:

```
UPDATE študent
SET id_izba = 4
WHERE id =104;
```

sa systém oprávnene ohradí:

```
... Application raised error with diagnostic text:
"Maj rozum! To má spať na zemi?".
SQLSTATE=HUSTO
```

A.2 Zdroje

- E. F. Codd: A Relational Model of Data for Large Shared Data Banks, Communications of the ACM (Vol. 13, No. 6, June 1970, pp. 377 – 387)
- DB2 help, `http://publib.boulder.ibm.com/infocenter/db2help/index.jsp`
- M. Kocan: Databáze nejsou jen relační, séria článkov, `http://www.dbsvet.cz`
- J. Pokorný, I. Halaška: Databázové systémy, Vybrané kapitoly a cvičení, Karolinum, Nakladatelství Univerzity Karlovy, Praha, 1994, ISBN 80-7184-687-2
- J. Pokorný: Databázová abeceda, Science, Veletiny, 1998, ISBN 80-86083-02-0
- R. Soták, rukopis prednášky
- J. D. Ullman: Principles of Database and Knowledge-Base Systems, Volume I., Computer Science Press 1988, ISBN 0-7167-8158-1
- J. D. Ullman: Principles of Database and Knowledge-Base Systems, Volume II., Computer Science Press 1989, ISBN 0-7167-8162-X
- J. Vinař, poznámky z prednášky

A.3 Index prvkov SQL

|, 26
ADD, 102
ADD COLUMN, 102
ALL, 133
ALTER, 105
ALTER COLUMN, 105
ALTER TABLE, 102
AND, 44
ANY, 133
ASC, 20
AVG, 48
BACKUP DATABASE, 186
BEGIN ATOMIC, 179
BETWEEN, 42
BIGINT, 16
BIND, 188
BLOB, 16
CASE, 35
CAST, 29
CHAR, 16
CHAR VARYING, 15
CHARACTER, 16
CHARACTER VARYING, 15
CHECK, 78
CLOB, 16
CLOSE, 189
CLUSTER, 181
COALESCE, 32
CONCAT, 26
CONNECT, 186
CONNECT RESET, 186
CONSTRAINT, 105
COUNT, 46
CREATE DATABASE, 186
CREATE FUNCTION, 190
CREATE INDEX, 181
CREATE TABLE, 15
CREATE TRIGGER, 174
CREATE VIEW, 165
CUBE, 50
CURRENT DATE, 68
DATE (dátový typ), 16
DATE (funkcia), 38
DAY (pre dátum), 33
DAY (pre rozdiel dátumov), 38
DAYNAME, 34
DAYOFWEEK, 33
DAYOFYEAR, 34
DAYS, 37
DEC, 16
DECIMAL, 16
DECLARE CURSOR FOR, 189
DECLARE SECTION, 189
DEFAULT, 67
DELETE, 66
DESC, 20
DISCONNECT, 186
DISTINCT, 47
DOUBLE, 16
DROP FUNCTION, 190
DROP DATABASE, 186
DROP INDEX, 181
DROP TABLE, 71
DROP TRIGGER, 177
DROP VIEW, 173
EMB_SQL_CHECK, 189
ESCAPE, 43
EXCEPT, 130
EXEC SQL, 189
EXISTS, 134
FETCH INTO, 189
FOREIGN KEY, 95
FULL OUTER JOIN, 91
GRANT, 186
GROUP BY, 49
HAVING, 52
IN, 42
INCLUDE SQLCA, 189
INSERT, 16
INT, 16
INTEGER, 16
INTERSECT, 130
IS NOT NULL, 41
IS NULL, 41
JOIN, 88
LCASE, 28
LEFT OUTER JOIN, 89
LENGTH, 28
LIKE (kópia), 60
LIKE (porovnanie), 42
LOWER, 28
MAX, 48
MIN, 48
MONTH (pre dátum), 33
MONTH (rozdiel dátumov), 38
MONTHNAME, 35
NOT, 44
NOT NULL, 73
NULL, 16
NULLIF, 41
NUM, 16
NUMERIC, 16
ON DELETE CASCADE, 97
ON DELETE SET NULL, 99
OPEN, 189
OR, 44
ORDER BY, 18
PREP, 188
PREPARE FROM, 190
PRIMARY KEY, 72
REAL, 16
REFERENCES, 94
RENAME TABLE, 122
RESTORE DATABASE, 187
REVOKE, 186
RIGHT OUTER JOIN, 90
ROUND, 31
RTRIM, 30
SELECT, 18
SET, 175
SET DATA TYPE, 105
SIGNAL SQLSTATE, 175
SMALLINT, 16
SOME, 133
SQLCODE, 189
SUBSTR, 27
SUM, 48
TIME, 16
TIMESTAMP, 16
UCASE, 28
UNION, 129
UNION ALL, 126
UNIQUE, 76
UPDATE, 62
UPPER, 28
USING, 190
VALUE, 33
VALUES, 21
VARCHAR, 15
WHERE, 39
WITH, 140
WITH CHECK OPTION, 171
WITH DEFAULT, 67
WITH GRANT OPTION, 186
YEAR (pre dátum), 33
YEAR (pre rozdiel dátumov), 38

A.4 Index pojmov

1:1, 111
1:n, 109
1. normálna forma, 223
1NF, 223
2. normálna forma, 225
2NF, 225
3. normálna forma, 226
3NF, 226
4. normálna forma, 229
4NF, 229
abstraktná tabuľka, 192
agregácia, 46
agregácia, marginálna, 50
agregačná funkcia, 46
akcia, 174
akcia, atomická, 179
algebra, relačná, 210
alias stĺpca, 23
alias tabuľky, 55
ANSI, 11
arita, 196
aritmetická funkcia, 29
Armstrongove pravidlá, 218
atomická akcia, 179
BCNF, 227
bezstratová dekompozícia abstraktnej tabuľky, 233
bezstratová dekompozícia schémy, 236
Boyceova-Coddova normálna forma, 227
britva, Occamova, 222
byť funkčne závislý, 217
byť multizávislý, 228
byť závislý, 217
Codd, Edgar Frank, 10
cudzí kľúč, 94
číselná funkcia, 29
číselník, 84
člen, neurčitý, 108
člen, určitý, 108
databáza, 12
databázová technológia, 12
databázový model, hierarchický, 10
databázový model, objektový, 10
databázový model, relačný, 10
databázový model, sieťový, 10
databázový model založený na správe súborov, 10
databázový systém, 12
dáta abstraktnej tabuľky, 192
dáta reálnej tabuľky, 194
dáta typovanej relácie, 194
dátový typ, 192
dátový typ stĺpca, 192
dátový typ údajov, 15
DB2, 10
dedič, 111
deklaratívny jazyk, 12
dekompozícia (Armstrongovo pravidlo), 218
dekompozícia abstraktnej tabuľky, bezstratová, 233
dekompozícia relácie typu m:n, 110
dekompozícia schémy, bezstratová, 236
dialekt SQL, 11
dopyt (príkaz), 18
dopyt (všeobecne), 12
dopyt, vnorený, 53
dopytovací jazyk, 12
dotaz, 12
druhá normálna forma, 225
duplicita dát, ne-, 12
dynamické embedded SQL, 189
Edgar Frank Codd, 10
elementárna funkčná závislosť, 217
elementárna T-podmienka, 198
embedded SQL, 188
embedded SQL, dynamické, 189
embedded SQL, statické, 189
entita, 108
entitno-relačný model, 112
entitný typ, 108
existenčný kvantifikátor, 133
forma, normálna, 223
forma, normálna, Boyceova-Coddova, 227
forma, normálna, druhá, 225
forma, normálna, prvá, 223
forma, normálna, štvrtá, 229
forma, normálna, tretia, 226
FoxPro, 10
funkcia, agregačná, 46
funkcia, aritmetická, 29
funkcia, číselná, 29
funkcia, definovaná používateľom, 190
funkcia, používateľom definovaná, 190
funkcia, typovaná, 196
funkčná závislosť (definícia), 217
funkčná závislosť (všeobecne), 122
funkčná závislosť, elementárna, 217
funkčná závislosť, redukovaná, 222
funkčná závislosť, redundantná, 221
funkčná závislosť, triviálna, 217
funkčne určovať, 217
generalizácia, 111
generalizácia, úplná a disjunktná, 111

hierarchia, IS-A, 111
hierarchický databázový model, 10
hodnota, prázdna, 16
hodnota, neutrálna, 68
hodnota termu, 198
hostiteľský jazyk, 190
chlpatosť, ne-, 224
idea, platónovská, 108
identifikátor objektu, 108
identifikátor záznamu, 71
index, 181
index, zhlukový, 181
indukcia, matematická, 156
Informix, 10
Ingres, 10
inštancia, 108
integritné obmedzenie, 12
IS-A hierarchia, 111
ISO, 11
Jan Vinař, 223
jazyk, deklaratívny, 12
jazyk, dopytovací, 12
jazyk, hostiteľský, 190
jazyk, procedurálny, 12
kanonické pokrytie, 221
kardinalita, 112
karteziánsky súčin, 207
kaskádové mazanie, 99
kľúč, 72
kľúč, cudzí, 94
kľúč, primárny, 72
kľúč, sekundárny, 72
kľúč schémy, 223
kľúčový stĺpec, 223
kompozícia, 218
konceptuálne modelovanie, 112
konkatenácia, 203
konštanta, 197
kontrola, 78
kurzor, 189
kvalifikovaný názov stĺpca, 85
kvantifikátor, existenčný, 133
kvantifikátor, všeobecný, 133
lepidlo, 26
logická schéma, 12
logika, trojhodnotová, 40
ľavé vonkajšie spojenie tabuliek, 89
ľavé vonkajšie spojenie typovaných relácií, 203
m:n, 110
marginálna agregácia, 50
maska, 42
matematická indukcia, 156
mazanie, kaskádové, 99
metadáta (dáta v systémových tabuľkách), 183
metadáta (charakter dát), 12
metadáta abstraktnej tabuľky, 192
metadáta reálnej tabuľky, 194
metadáta typovanej relácie, 193
metamodel, 183
množina, 108
množina stĺpcov, redukovaná, 222
model, databázový, hierarchický, 10
model, databázový, objektový, 10
model, databázový, relačný, 10
model, databázový, sieťový, 10
model, databázový, založený na správe súborov, 10
model, entitno-relačný, 112
modelovanie, konceptuálne, 112
MS Access, 10
MS SQL Server, 10
multiurčovať, 228
multizávislosť, 228
multizávislosť, triviálna, 228
MySQL, 10
nadkľúč, 223
názov stĺpca, 192
názov stĺpca, kvalifikovaný, 85
ne-chlpatosť, 224
neduplicita dát, 12
nechlpatosť, 224
neredundantnosť dát, 12
neurčitý člen, 108
neutrálna hodnota, 68
normalizácia, 85
normálna forma, 223
normálna forma, Boyceova-Coddova, 227
normálna forma, druhá, 225
normálna forma, prvá, 223
normálna forma, štvrtá, 229
normálna forma, tretia, 226
nullabilita stĺpca, 192
nullabilita termu, 197
nullabilitné pravidlo, 196
nullabilitné správanie, 196
objekt (inštancia triedy), 108
objekt (jednotlivina), 108
objektový databázový model, 10
obmedzenie, integritné, 12
Occamova britva, 222
odpoveď, 18
Oracle, 10
parameter vnoreného dopytu, 55
perzistencia dát, 12
placatosť, 223
platónovská idea, 108
podmienka, 198

pohľad, 165
pohľad, rekurzívny, 168
pojem, 108
pokrytie, kanonické, 221
pokrytie množiny závislostí, 221
pokrytie množiny závislostí, kanonické, 221
položka záznamu, 16
pomocná tabuľka, 140
PostgreSQL, 10
postupnosť, vytvárajúca, 209
používateľom definovaná funkcia, 190
pravé vonkajšie spojenie tabuliek, 90
pravé vonkajšie spojenie typovaných relácií, 203
pravidlá, Armstrongove, 218
pravidlo, 108
pravidlo, nullabilitné, 196
prázdna hodnota, 16
prázdnaschopnosť stĺpca, 192
premenná, 197
premenovanie stĺpcov, 202
prienik tabuliek, 130
prienik typovaných relácií, 208
primárny kľúč, 72
prirodzené spojenie, 230
procedúra, 55
procedurálny jazyk, 12
projekcia (ako operácia relačnej algebry), 202
projekcia abstraktnej tabuľky, 231
projekcia množiny závislostí, 235
projekcia schémy, 235
prvá normálna forma, 223
prvok, 108
query, 12
reálna tabuľka, 194
redukovaná funkčná závislosť, 222
redukovaná množina stĺpcov, 222
redukovaná závislosť, 222
redundantná závislosť, 221
redundantnosť dát, ne-, 12
redundantný stĺpec, 222
rekurzia, 156
rekurzívny pohľad, 168
relačná algebra, 210
relačný databázový model, 10
relácia, typovaná, 193
relácia typu 1:1, 111
relácia typu 1:n, 109
relácia typu m:n, 110
riadok abstraktnej tabuľky, 192
riadok typovanej relácie, 194
rola, 111
rozdiel tabuliek, 130
rozdiel typovaných relácií, 208
rýchlosť prístupu k dátam, 12
samovzťažnosť, 185
sekundárny kľúč, 72
selekcia riadkov, 203
selekt, 18
schéma, 185
schéma abstraktných tabuliek, 219
schéma, logická, 12
sieťový databázový model, 10
skupina, 49
skupinovací výraz, 49
spojenie, prirodzené, 230
spojenie schém, 235
spojenie tabuliek, 87
spojenie tabuliek, ľavé vonkajšie, 89
spojenie tabuliek, pravé vonkajšie, 90
spojenie tabuliek, úplné vonkajšie, 91
spojenie tabuliek, vnútorné, 93
spojenie tabuliek, vonkajšie, ľavé, 89
spojenie tabuliek, vonkajšie, pravé, 90
spojenie tabuliek, vonkajšie, úplné, 91
spojenie typovaných relácií, 203
spojenie typovaných relácií, ľavé vonkajšie, 203
spojenie typovaných relácií, pravé vonkajšie, 203
spojenie typovaných relácií, úplné vonkajšie, 203
spojenie typovaných relácií, vnútorné, 203
spojenie typovaných relácií, vonkajšie, ľavé, 203
spojenie typovaných relácií, vonkajšie, pravé, 203
spojenie typovaných relácií, vonkajšie, úplné, 203
spoľahlivosť dát, 12
správanie, nullabilitné, 196
spúšťač, 174
SQL, 11
SQL89, 11
SQL92, 11
SQL99, 11
SQL, embedded, 188
SQL, embedded, dynamické, 189
SQL, embedded, statické, 189
statické embedded SQL, 189
stĺpec, 192
stĺpec, kľúčový, 223
stĺpec, redundantný, 222
Structured Query Language, 11
superakcia, 179
súcno, 108
súčin, karteziánsky, 207
Sybase, 10
systém, databázový, 12
systém riadenia databázy, 12
systémová tabuľka, 183
špecializácia, 111
štvrtá normálna forma, 229

T-termy, 197
T-podmienka, 198
T-podmienka, elementárna, 198
T-premenná, 197
tabuľka, 12
tabuľka, abstraktná, 192
tabuľka, pomocná, 140
tabuľka, reálna, 194
tabuľka, systémová, 183
technológia, databázová, 12
term, 197
tica, 40
transakcia, 179
transformácia stĺpcov, 201
tranzitivita (relácie), 156
tranzitivita (Armstrongovo pravidlo), 218
tranzitívny uzáver, 156
tretia normálna forma, 226
trieda, 108
trigger, 174
triviálna funkčná závislosť, 217
triviálna multizávislosť, 228
trojhodnotová logika, 40
typ, 192
typ 1:1, 111
typ 1:n, 109
typ, dátový, 192
typ, entitný, 108
typ m:n, 110
typ stĺpca, 192
typ stĺpca, dátový, 192
typ termu, 197
typovaná funkcia, 196
typovaná relácia, 193
UDF, 190
určitý člen, 108
určovať, 217
user defined function, 190
uzáver množiny stĺpcov, 222
uzáver množiny závislostí, 220
uzáver, tranzitívny, 156
únikový znak, 43
úplná a disjunktná generalizácia, 111
úplné vonkajšie spojenie tabuliek, 91
úplné vonkajšie spojenie typovaných relácií, 203
view, 165
Vinař, Jan, 223
vnorený dopyt, 53
vnútorné spojenie tabuliek, 93
vnútorné spojenie typovaných relácií, 203
všeobecný kvantifikátor, 133
vyhodnotenie T-podmienky, 199
vytvárajúca postupnosť, 209
výraz, skupinovací, 49
wildcard, 42
závislosť, funkčná (definícia), 217
závislosť, funkčná (všeobecne), 122
závislosť, funkčná, elementárna, 217
závislosť, funkčná, redukovaná, 222
závislosť, funkčná, redundantná, 221
závislosť, funkčná, triviálna, 217
závislosť, redundantná, 221
záznam, 16
záznam abstraktnej tabuľky, 192
záznam typovanej relácie, 194
zhlukový index, 181
zhodovať sa, 217
zjednotenie tabuliek, 126
zjednotenie typovaných relácií, 207
zmysel pre symetriu, 148
zmysel pre zákony zachovania, 190
znak, únikový, 43
zovšeobecnenie, 111
žolík, 42