FORMALIZACIJA VEČJE KOLIČINE PODATKOV

Rado JENSTERLE

1. Uvod

Če imamo veliko sinonimov pri *imenih* (različna imena za isti objekt), potem jih sistem oz. aplikacija med seboj ne razlikujeta. Tako se lahko zgodi, da vodimo v registru 100.000 različnih oseb, zapisanih pa je 500.000, ker se npr. ista oseba v registru pojavi v povprečju petkrat. Posledice so zelo odvisne od aplikacije, večinoma pa so močno nezaželjene.

Podoben primer tvorijo slabo oblikovani *naslovi*. Naslovi so po naravi precej redundantni, saj na istem naslovu lahko živi večje število subjektov. Za razliko od osebnih imen, ki so praviloma konstante, pa se naslovi kar pogosto menjavajo in je doslednost njihovega pisanja zelo pomembna. Stvar je toliko bolj pomembna, kolikor je aplikacija odvisna od prostorske topologije v kateri se subjekti nahajajo (npr. marketing, spremljanje množice kupcev ipd.).

Današnja informatika mora biti vsklajena z zakonodajo, zato ni zakonito niti modro, da bi enolično identifikacijo oseb zasnovali na neki splošni enotni matični številki osebe (npr. EMŠO). Izkušnje nas uče, da ime in priimek ne zagotavljata zadostne enoličnosti za vsakodnevne potrebe. Lepo bi bilo, da je vsak posameznik čim bolj enolično določljiv znotraj nekega podjetja (npr. 99%), vendar pa želimo zaščititi njegove osebne podatke in ne težimo k temu, da bi se naš interni sistem povezoval z nekim drugim sistemom (predmet zaupanja). Enolično identifikacijo želimo izboljšati z večjo formalizacijo imen in naslovov in eventuelno temu prilagojenim sistemom šifriranja. Glavni problem pri dosegu tega cilja pa tiči v neurejenosti starih, obstoječih podatkov.

Pričujoči prispevek bo govoril o uspešni formalizaciji več 100.000 osebnih imen z njihovimi naslovi, pri čemer nas je vodilo pravilo: " Za formalizacijo uporabljamo samo legalne in javne osebne podatke, katere nam zaupa stranka sama". Izrazit napor je bil vložen v formalizacijo naslovov, ki imajo v praksi izredno širok spekter uporabe, pa istočasno pomenijo največji filter za možne sinonime kljub spremenljivosti. Predstavili smo kombinacijo vertikalnega in horizontalnega očiščevanja podatkov z uporabo jezika SQL, pri čemer jezik ni osrednja tema prispevka.

2. Filtriranje podatkov

Za primerno očiščevanje podatkov moramo najprej izbrati ustrezne filtre. Filtriranje v običajni praksi pomeni odstranitev (ločitev) nečistega dela od čistega. V informatiki si vedno tega ne moremo privoščiti, saj ne moremo posameznikov, ki imajo neformalen zapis imena ali naslova, izločiti. Zato gre mnogokrat za ločevanje in popravljanje nečistega in le izjemoma za odstranjevanje. Če je odstranjevanje dopustno, je naloga mnogo lažja, le granulacijo nečistoče, torej mejo med čistim in nečistim, je včasih težko določiti. V primeru ko smemo nečiste podatke le popraviti, imenujmo nečisti podatek raje neformalen.

Neformalni podatki v glavnem pomenijo večja ali manjša oblikovna odstopanja od znanih ali željenih oblik. Primeri variacij tipičnih *osebnih imen* in *naslovov*:

Janez Klepec	Gorenjska c. 3
dr. Janez Jože Klepec	Gorenjska cesta 3
Janez Klepec dipl. ing.	Gor. cesta III 22
Janez Klepec-Kosec dipling.	Gorenjska c 22/XIX-2103
prof. Janez Kosec Klepec dipl iur	Gorenjska n.h.
ga. Janja Kosec roj. Klepec	Gorenjska ul. st. 5-3 nad.-soba 210
itd.	ipd.

Možne so razne kombinacije zgoraj nakazanih variant, pri čemer so dodatno možne vsebinske napake kljub ustreznemu formalizmu. Pri naslovom je potrebno vračunati še zmešnjavo pri *krajih, naseljih, poštah* in *občinah*, ki jih spremljajo. Podobna zmešnjava kot pri osebnih imenih vlada tudi pri imenih pravnih oseb, zlasti zato ker so ključni deli imen pravnih oseb nekje v sredini imena. Izkušnja kaže, da je število odstopanj skoraj nemogoče oceniti, ter da se število netipičnih vzorcev giblje od 50 do 100 po vrsti objekta.

3. Programerske tehnike filtriranja

Klasični *programerski* "IF..." princip odpove, ker je spekter posebnih primerov tako širok, da bi klasično programiranje pri veliki množici podatkov lahko trajalo tudi 1 leto, če bi hoteli sproti obdelovati nove in nove nastale neformalne vzorce. Večinoma velja pravilo, da pripada zelo specifičnim vzorcem, še zlasti ko gre za napake, le majhna množica reprezentančnih podatkov. Programiranje filtrov bi se morda izplačalo le v primeru, kjer imamo jasno definirano mejo tolerance in vse kar je izven tolerance zavržemo. Kot rečeno, večinoma tega ne moremo storiti, saj mora npr. banka skrbeti prav za vsakega svojega komitenta. Za povezanost poslovanja je včasih bolje imeti istega komitenta petkrat v podatkovni bazi, kot pa da bi ga zaradi nepredvidljivosti postopka zavrgli.

Principi, ki jih omogoča relacijska baza z uporabo SQL jezika, pa so popolnoma drugačni. SQL je namenjen delu z množicami in ne le s posameznimi njenimi elementi. Zato pri SQL

množico najprej napademo vertikalno, to pomeni, da s posebnimi funkcijami na nivoju množice najprej iščemo podmnožico, ki ustreza kakšnemu znanemu vzorcu. Bistveno pri tem je, da vzorce šele iščemo, kar nam zopet omogoča univerzalnost jezika SQL. Dejstvo, da nam ni vnaprej potrebno poznati vseh (ne)formalnih vzorcev, daje bistveno prednost uporabi strukturnega jezika pred klasičnim proceduralnim jezikom. V praksi sem dosegel popolno očiščenje zgoraj omenjenih podatkov brez enega samega klasičnega programskega ukaza, celo vse v interaktivnem načinu dela. Ko je vzorec znan, se s SQL lotimo ustrezne podmnožice posameznih primerov, ki jih potem sistematično obdelamo. Če imate nekaj izkušenj, vam SQL omogoča tudi to.

4. Metoda filtriranja ali metoda zgoščevanja

Bistveni predpostavki za uporabo moje metode sta vključitev vseh primerkov in vnaprej nepoznan *vzorec* formalizma. Oba kriterija, torej *številčnost* in *raznolikost* pa sta med seboj bolj ali manj povezana po pravilu 80:20. Znano pravilo pove, da iščemo najprej vzorce, ki družijo veliko število primerkov in takih vzorcev je malo, tako da z njimi pokrijemo 80% populacije. Isto pravilo pove, da nam na koncu ostane veliko število vzorcev z majhnim številom primerkov, ki vsi skupaj pokrivajo 20% populacije. Razmerje 80% in 20% je ilustrativno in seveda v praksi odstopa, pa vendar ga izkušnje potrjujejo. Čisto nazadnje nam na dnu filtra ostane *usedlina*, kjer se moramo ukvarjati individualno s posameznimi primerki, ki pa jih je relativno malo celo za ročno popravljanje. Tudi usedlina je pomembna, saj so v njej v glavnem skriti podatki z napakami. Razen tega nam nudi enkratno priložnost za odpravo temeljnih podatkovnih nečistoč. Včasih pa si lahko privoščimo, da "zoc" kar zavržemo.

Vključitev vseh primerkov pomeni, da bomo imeli veliko število reprezentančnih vzorcev, zato je temeljnega pomena iskati kriterije za zgoščevanje populacije. Kriterij, ki zajame reprezentančni vzorec populacije imenujmo kar *vzorec filtra*. Prvi vzorci filtra, ki zajemajo visoke populacije, nas tudi pripeljejo do bodočih *formalizmov*.

Posebna umetnost je iskanje formalizmov. Neformalen podatek je omejen edino s podatkovnim tipom, npr. dolžina, numeričen, alfabetski, posebni znak ipd. Druga skrajnost formalizma nam je poznana iz rešitev za osebne računalnike, kjer smemo izbirati prek izbirnega okna samo natančno določeno vrednost (šifranti), kjer ni prav nobene svobode. Stvarnost velikih podatkovnih zbirk pa je nekje vmes, saj bi radi zajeli veliko število primerkov, pri čemer pa bi obliko omejili na natančno določeno število formalizmov (npr. 10) in dovolili uporabnikom oz. programerjem, da se sami odločajo, katere med njimi bodo uporabili na nekem primerku. Te formalizme pa je najbolje določiti iz populacije, saj ne obstajajo neki standardi, ki bi nam tu olajšali delo. Vsekakor pa taki formalizmi vodijo v neke bodoče *standarde*.

Metoda filtriranja nam torej omogoča z vzorci filtrov obdelati masovni del populacije. Ostanek neobdelane populacije se vedno bolj zgoščuje, v njem so v veliki meri skriti podatki, ki poleg posebnih formalizmov vsebujejo predvsem napake. Metoda nas tako vodi do posamične obravnave primerkov na relativno zmanjšani populaciji. Na koncu imamo vse podatke očiščene, natančno identificirane vse vzorce formalizmov, statistično porazdelitev populacije nad posameznimi vzorci, kar nas lahko pripelje do odločitev, ki vpeljejo neko obliko standardizacije formalizmov. *Standardne vzorce* bo v bodoče možno obvladati tudi s klasičnimi programerskimi tehnikami, kljub temu, da bomo uporabnikom in programerjem dovolili nekaj svobode pri delu. Odstopanja od standardnih vzorcev je možno obvladovati z občasnimi paketnimi obdelavami.

5. Praktične izkušnje prečiščevanja podatkov

V praksi se iskanje tipičnih vzorcev prevede na iskanje razlik in podobnosti, in sicer čimveč podobnih značilnosti na populaciji najprej, manjše populacije pa čim kasneje. Iskanje neke zakonitosti na vsakem vzorčnem reprezentantu je horizontalni vidik, številčnost populacije vzorca pa je *vertikalni vidik* nekega problema. Vertikalne vidike rešujemo v SQL s funkcijami tipa GROUP BY, HAVING, DISTINCT, MAX, MIN, SUM, COUNT itd., horizontalni vidiki se rešujejo s primerno klavzulo WHERE, ki določa populacijo vzorca, in skalarnimi funkcijami nad elementi stavka SELECT. Vertikalni vidik zahteva od "čistilca" veliko domišljije in poznavanja baze s performančnega vidika, *horizontalni* vidik pa izjemno veliko izkušenj in predstave, če želimo obvladati čiščenje brez dodatnega programiranja. Ena od tipičnih nalog je npr. razbiti osebno ime, ki se sestoji iz treh ali več neformalnih sestavin in jih nazaj sestaviti (ali pa tudi ne) v pravem zaporedju. Domača naloga!

Potem, ko je vzorec določen, je potrebno na celotni podpopulaciji izvesti ustrezne popravke in spremembe. Poleg že omenjene akcije na vsakem primerku, se tu pokaže kot dodaten problem struktura vzorca. *Struktura vzorca* včasih od nas zahteva več korakov v horizontalni akciji, torej rekurzivno reševanje. Poseben problem je natančno določevanje spola pri osebnih imenih, kar je možno le do zelo visokega procenta (nad 99% populacije 100% natančno). V praksi to pomeni, da si pripravimo posamične *vzorčne elemente*, ki jih potem uporabimo na vzorcu v več korakih z uporabo tehnik JOIN. Problem takega združevanja dveh tabel pa je v tem, da zahteva med primerki obvezni operator "=" . Ta omejitev pa nam bistveno jemlje maneverski prostor pri delu z "različnimi" stvarmi. Dodatni problem je performančne narave, saj delo s podnizi (funkcija SUBSTRING) izloča uporabne indekse in naletimo lahko na nesprejemljive performančne težave. Ravno tukaj je potrebno dobro poznavanje SQL in vpliva tega jezika na performanse.

Izkušnja je pokazala, da je kljub zapletom, ki jemljejo čas, možno priti do popolnega rezultata, in sicer bistveno hitreje kot s klasičnim programiranjem, kadar je kvaliteta podatkov pomembna. SQL se spoprime prav z vsakim primerkom na populaciji. Potrebna je le doslednost v dolgem nizu korakov od začetka do konca. Ker vsaka logična napaka pomeni ustvarjanje novih nečistoč, je s postopkom prečiščevanja potrebno stalno vzporedno graditi tudi kontrolni postopek.

6. Statistika

V praksi sem izpeljal tako nalogo na populacij 500.000 primerkov v okolju SQL/DS. Ena tabela je zasedala približno 100 MB prostora, potrebno pa je bilo kar nekaj takih tabel, da sem lahko shranil vse vmesne rezultate. Bistven del problema so bili poleg imen naslovi, ki jih je bilo potrebno uskladiti z občinskimi, poštnimi in statističnimi šifranti.

Nalogo sem izpeljal prek približno 80 vmesnih delovnih tabel različnih velikosti, kar je zajemalo več kot 100 postopkov v SQL.

Nekateri ukazi so bili izjemno zahtevni, saj se je en sam ukaz raztezal prek dveh ekranov. Eden od teh ukazov je bil logično tako zapleten, da sem potreboval tri dni testiranja, predno je deloval. Tukaj bi s klasičnim programiranjem lahko dosegel isti rezultat v enem dnevu. Kasneje pa je ta ukaz doživel še mnogo variacij, za katere sem potreboval le nekaj minut, v klasičnem programu pa bi tu izgubljal ure in dneve. Večina končnih postopkov so bili modificirani osnovni postopki, tudi v primeru tistega zapletenega.

Zelo pomemben je performančni vidik prečiščevanja. Delo s tako velikimi tabelami zahteva stalno stoodstotno osveščenost dogajanja v sistemu. Delo na velikih vzorcih oz. celi tabeli je v povprečju zahtevalo pol ure do 45 minut CPU za en postopek. Takih postopkov je bilo okoli 100. Kasneje se je pokazalo, da je isti postopek v OS/2 DB2 zahteval nekaj ur dela, v okolju PC DOS BTRIVE pa nekaj dni. Pri delu z osebnim računalnikom je performančna komponenta še toliko bolj pomembna.

Zaključek

Kljub težavam je bilo celotno delo zaključeno v 40 dneh, v kar je vštetih mnogo dni ročnega popravljanja podatkov iz usedline. V te dneve je všteta kompletna izdelava orodja za prečiščevanje podatkov. Na koncu sem porabil največ časa za čakanje rezultatov posameznih korakov, ki pa sem ga uspešno izkoristil za opravljanje vzporedne kontrole kvalitete opravljenega dela. Napak skorajda ni bilo. Rezultat dela je enotni interni register strank z možnostjo dodatne kontrole prek naslovov, ki so urejeni v okviru ažurne informacije in vsklajeni z drugimi obstoječimi standardi v naši državi.

Na posvetovanju Dnevi slovenske informatike 95' v Portorožu je avtor prejel priznanje udeležencev.

Rado Jensterle je zaposlen pri podjetju Genis d.o.o., Ljubljana

■

"POGOVARJANJE" V RAČUNALNIŠKIH OMREŽJIH

Zenel Batagelj

1. Uvod

Razvoj računalniških omrežij je prinesel nov način uporabe računalnikov tj. v komunikacijske namene. Nastala je skovanka računalniško posredovane komunikacije (*Computer Mediated Communication - CMC*), kjer se je med besedi računalniške in komunikacije vrinila beseda posredovanost, ki poudarja, da ne gre več za komuniciranje med računalniki ampak za uporabo računalnikov za medčloveško komunikacijo. Poudariti je treba, da se oblike CMC med sabo zelo razlikujejo. Pod CMC tako sodijo vse oblike medosebnega komuniciranja kot tudi vse oblike doseganja informacij (World Wide Web, Gopher...). Osredotočili se bomo na medosebno komuniciranje.

Obstajata dva osnovna kriterija delitve. Prvi kriterij je sinhronost. Gre za vprašanje, koliko časa mine med oddajanjem in sprejemanjem sporočila. Glede na ta kriterij ločimo asinhrone (elektronska pošta) in sinhrone (IRC, VAX phone) komunikacije. Drugi kriterij pa je število komuncirajočih, pri čemer so seveda najbolj zanimive oblike komunikacij z več kot dvema komuncirajočima - najboljši primer so konferenčni sistemi (USENET, ADRIANET).

2. Jezikovna komunikacija

Jezikovna komunikacija običajno poteka po dveh prenosnikih: govornem in pisnem. Pri računalniško posredovanih komunikacijah[1] poteka komunikacija izključno prek pisnega prenosnika - prek znakov na zaslonih. Točneje, komunikacija poteka prek tipkovnic, kar pomeni, da smo pri izbiri znakov omejeni na tiste, ki so dosegljivi prek tipkovnice tj. na velike in male črke abecede in pa posebne - tipografske znake.

Tomo Korošec (1986) je analiziral različne situacije pri komuniciranju prek obeh prenosnikov:

I. **pisni prenosnik**:
   - A. besedila namenjena branju; praviloma se ne berejo glasno (čista varianta),
   - B. besedila, ki se glasno berejo (recital), vendar se samo besedilo ne razlikuje glede na točko a,
   - C. besedilo, pri katerem se že v fazi oblikovanja upošteva, da bo posredovano prek govornega prenosnika;

II. **govorni prenosnik**:
   - A. besedila namenjena izključano slušnemu sprejemanju (čista varianta):
     1. pogovor, kjer se komunicirajoči vidijo (*face to face*),
     2. pogovor, kjer se komunicirajoči ne vidijo (prek telefona);
   - B. prenos nekega pogovora, kjer se komentira dejavnost udeležencev komunikacije,
   - C. govorjenje, katerega namen je, da se transformira v pisni prenosnik.

Z vidika *CMC* je zgornja delitev zelo pomembna, saj komunikacija poteka le prek pisnega prenosnika kot njegova čista varian-

---

1 *Do leta 1993, ko so pri NCSA naredili prvi grafični vmesnik Mosaic za WWW, je tudi v informacijskih prostorih komunikacija potekala le prek pisnega prenosnika. Danes pisni prenosnik dopolnjujejo slike, film in vse bolj tudi zvok.*