ERM comparing - pár slov k implementácii

Adam Trizna

09.08.2020

1 Úvod

Tento text by mal byť nejaké stanovisko ako výsledok doterajších úvah. Ide o to, ako implementačne poňať rozširovanie backtrackingového stromu (cez transformácie modelov) počas hľadania optimálneho mapovania. Účel je, aby to bolo spísané a nemusel by som sa k tomu už vracať.

Hovorím tu len o základnej NEoptimalizovanej verzii. Prípadné rezanie, paralelizmus, iná mágia, .. príde neskôr.

Popritom všetkom naďalej platia pôvodné myšlienky, že popri vykonávaní transformácií sa mapujú entitySety naprieč modelmi, čo tuto slúži ako "fixovanie entiySetov", tj. zafixované entitySety už nemožno použiť ako vstup do nejakej transformácie.

2 Scenár 1

Prvý scenár je implementačne ďaleko náročnejší. Ide o to, že by si počas backtracku uvažoval naozaj všetky možné aplikovateľné transformácie, aké existujú (toto teda bola tá pôvodná najčistejšia zamýšľaná forma, teda že výsledok je presná postupnosť transformácií, ktorými z jedného modelu dostaneš druhý).

Myslím tým to, že v konkrétnom node bactrack stromu by si zozbieral všetky môžné druhy transformácií (komplexné (všetky) aj jednoduché (vrátane flipovania kardinality, pridania/odstránenia atribútu/entitySetu, ..)), zistil by si, pre aké všetky vstupy je možné transformácie vykonať, následne ich v rámci backtracku vykonal.

Tento scenár tiež potrebuje nejakú veľmi netriviálnu logiku na obmedzenia "možných transformácií", keďže veľmi ľahko sa môže stať, že sa to bude nejakým spôsobom cykliť, resp. na jednej vetve sa budú diať opačné transformácie, jednoducho že faktor rozvetvenia stromu by sa v konečnom dôsledku nezmižoval, čo by bolo nedobre. Predstavujem si to napríklad nejako tak, že by bolo treba strážiť, že každá kardinalita sa flipne len raz, alebo že je možné do modelu pridať najviac toľko entitySetov, koľko je v opačnom modeli, inak už nemajú zmysel (a kopec ďalších obmedzení).

Ak by boh dal a nejaká vetva backtrack stromu by sa dostala až na svoj koniec, s kľudom môžem zvoliť takú nejakú penalizáciu, že ak NIE sú aktuálne modely v mapovaní izomorfné, je to penalizácia za nekonečno (zamietam vetvu), v opačnom prípade je to penalizácia za konkrétne použité transformácie.

2.1 Výhody

Asi jediná výhoda tohto tu je, že by som sa nemusel báť, že by mi nejaké možné riešenie ušlo.

2.2 Nevýhody

Hlavnou nevýhodou je, že sa bavíme o úplne hrôzostrašne nemysliteľných číslach. Pre predstavu, uvažujme dva modely po 10 entitySetoch (čo ani nieje najviac, s čím sa stretnem), po dvoch atribútoch a po 10 binárnych vťahoch (čo by mohol byť nejaký priemer, práve som si to vymyslel). Uvažujme:

transformácia	približný počet vstupov
pridaj ES	1
odstráň ES	1
flipni kardinalitu	20
pridaj atribút	10
odstráň atribút	10
odstráň vzťah	10
pridaj bin. vzťah	45
komplexné úpravy (povedzme spolu)	10

Toto bol len hrubý odhad, ale dostatočne realistický. Môžem si teda predstaviť nejaký faktor rozvetvenia stromu približne 107. Zároveň dĺžka jednej vetvy sa pohybuje medzi 10 až (10 (za mapovanie) + $\geq$ 107 za transformácie

(ak sa to nijako nezacyklí)). **Toto bude menej preferované riešenie.**

3 Scenár 2

Druhý scenár vychádza z prvého, ale snaží sa ušetriť nejakú robotu.

V tomto prípade by som samostatne neuvažoval o jednoduchých transformáciach. Prečo sme toto celé (teda zakomponovanie transformácií do backtrack stromu) vymysleli je, aby sme vyskúšali exponenciálne veľa riešení s komplexnými transformáciami. Čo to znamená: o jednoduchých transformáciach by nám malo stačiť uvažovať len v dvoch rovinách:

1. Ako prípravné pre komplexné transformácie (ich aplikovaním by sa "odomkli" predispozície pre nejakú komplexnú transformáciu.

2. Ako samostatné transformácie (lebo aj také treba), v tejto podobe by ale malo stačiť ich vyčítať ako rozdiel na konci vetvy.

Čo si teda pod tým predstavujem: Vykoná sa backtrack, kde okrem mapovania robím komplexné úpravy. Zjavne by nestačilo iba zisťovať, aké komplexné úpravy je možné v danom node vykonať a vykonať ich. Dôvodom je, že predispozície nejakých komplexných úprav by platili až po vykonaní nejakých jednoduchých transformácií.

V takom prípade by bolo treba vymyslieť trochu inteligentnejšiu analýzu "aktuálne možno aplikovateľných transformácií" a to takú, ktorá by vrátila buď to holú komplexnú transformáciu (s konkrétnymi vstupmi) (ktorá je priamo aplikovateľná), alebo komplexnú transformáciu (s konkrétnymi vstupmi), ktorú nieje možné priamo aplikovať, a zoznam jednoduchých transformácií (s konkrétnymi vstupmi), po aplikácií ktorých bude možné aplikovať aj komplexnú transformáciu.

Čo sa týka penalizácie hotovej vetvy, nebude možné to robiť tak necitlivo ako v prvom scenári. Ako som už naznačil, tuto by sa z modelov a mapovania vyčítali jednoduché transformácie (toto je jednoduchý preklad), z ktorých by sa vypočítala penalizácia. To, či je možné z týchto jednoduchých transformácií poskladať nejakú komplexnú (ako som to zamýšľal dávnejšie pri prvých prístupoch) nás NEzaujíma. Platí nejaké heslo, že "pokiaľ chceš komplexnú transformáciu, urob si ju v strome".

3.1 Výhody

Určite ide o výpočtovo menej náročné riešenie (ako to prvé). Keďže v prvom prípade to bolo nemysliteľne náročné, **zjavne sa prikloním k tomuto**.

3.2 Nevýhody

Tuto nieje hneď len taká istota, že nájdem všetky riešenia (aj keď v tejto chvíli sa mi to pozdáva). Navrhujem prejsť príklady, ktoré máš, a overiť, že takýmto postupom by si sa dostal k cielenému výsledku.