Ročníkový projekt - report - letný semester

Úvod

Nadväzujúc na zimný semester, v letnom semestri bolo našim cieľom implementovať redukovanú sadu operátorov relačnej algebry (Join, AntiJoin a Union) v kóde. Pokúsili sme sa tiež o silnejšiu redukciu operátorovej sady na jeden všeobecný operátor, čo sa však v priebehu semestra ukázalo ako nie príliš veľká optimalizácia, relatívne k náročnosti implementácie takého operátora. Preto sme pokračovali s optimalizáciou sady troch operátorov, pričom sme sa zamerali na automatizáciu vytvárania konštruktorov pomocou loadera relačnej algebry a optimalizovali sme aj rekurzívne výpočty odstránením zbytočného Joinu pri volaní RecursiveOperator.instance().

Priebeh práce počas semestra

V minulom semestri spočívala práca najmä v návrhu redukovanej sady, tento semester najmä v jej programovaní:

1. implementácia n-árneho operátora AntiJoin
2. zmena konštruktora pre operátor Union (nový konštruktor už nemá obsahovať vstupný a výstupný vektor - ukázalo sa, že pre Union nemajú zmysel) a implementácia n-árneho operátora Union
3. úprava rekurzívneho operátora tak, aby sa v jeho metóde instance() nevolal zbytočný Join (uvedomili sme si, že Join v pôvodnej implementácii relačnej algebry iba porovnával dve tuple z dvoch úrovní rekurzívneho výpočtu a ak sa rovnali vrátil prvú, čo sa dalo efektívnejšie nahradiť overením, či je úroveň rekurzie (index U) párna a podľa toho vrátiť finálnu inštanciu operátorového stromu s konkrétnymi indexmi U a L). Tým sme z rekurzívneho stromu odstránili celú jednu vetvu výpočtu.
4. testy takto dokončenej operátorovej sady (najmä rekurzie)
5. implementácia loadera relačnej algebry pre účely automatizovania vytvárania konštruktorov v testoch (značné zjednodušenie vytvárania objektov triedy Operator)

Príklad 1 - loader

Majme binárne relácie r, s a datalogové pravidlo s dotazom:

p(A, B) ← r(A, B), s(A, B), ¬s(B, C).
? p(1, X)

Bez použitia loadera relačnej algebry by kód programu v Jave realizujúceho daný dotaz vyzeral nasledovne:

```
p = new AntiJoin(
        Arrays.asList(
                new Join(
                        Arrays.asList(r, s),
                        Arrays.asList(
                                Arrays.asList(A, B),
                                Arrays.asList(A, B)
                        ),
                        Arrays.asList(A, B)
                ), s),
        Arrays.asList(
                Arrays.asList(A, B),
                Arrays.asList(B, C)
        ),
        Arrays.asList(1, X)
);
```

Ten istý dotaz sa dá vďaka loaderu (ktorý prekladá textový vstup do objektov typu Operator) vyjadriť nasledovne:

```
p = loadOperator("ANTIJOIN(
        [ JOIN([ $r, $s ], [ [ A, B ], [ A, B ] ], [ A, B ]), $s ],
        [ [ A, B ], [ B, C ] ],
        [ 1, X ]
)");
```

Kde $r a $s sú značky používané ako odkazy do extenzionálnej databázy implementovanej ako objekt typu Database, ktorá obsahuje naplnené relácie r a s.

Prínos loadera sa ukázal hneď pri vytváraní testovacích inštancií, keďže umožňuje oveľa rýchlejšie a intuitívnejšie zapisovanie relačnej algebry.

Záver

Počas semestra sme už okrem vyššie spomenutého uvažovali aj nad uskutočnením materializácie (kde sa oplatí materializovať a čo všetko je potreba materializovať, keďže úplná materializácia je neperspektívna - možno by stačilo ukladať menšie množstvo medzivýsledkov rekurzie pre indexy U a L, ktoré sa v ďalších úrovniach rekurzívneho výpočtu ešte použijú). V tejto oblasti je však ešte veľa čo analyzovať, preto v prípade pokračovania v bakalárskej práci je tu priestor pre rozšírenie.