A Top-down Query Evaluation Engine for Datalog

Hana Sebia
Tarik Boumaza

Juin 2021

1 Objectif

Le sujet soumis à l'étude est l'implémentation d'un moteur d'évaluation de requêtes Datalog en Java.

2 Méthode

Structure du code

L'implémentation complète du programme d'évaluation de requête respecte le squelette fourni (permettant le parsing des fichiers "exemples.txt") et le raisonnement développé par le sujet.
Pour évaluer une requête à l'aide du moteur d'évaluation top-down (`QsqrEngine`), on effectue deux étapes : la génération du programme orné, puis l'évaluation récursive d'une requête orné.

Initialisation et pré-traitement

- *Adornement :*
  L'ornement est réalisé à l'aide de la classe `ArdornedRules` qui fournit une méthode `init()`, permettant de générer l'ornement d'un programme donné et de stocker les règles ornées. En effet, cette méthode génère d'abord l'ornement de la requête, avant de déterminer les règles permettant de définir les *IDB* présents dans son corps. Ensuite, elle lance un appel à la fonction récursive `RecursiveArdornedRules()` qui effectue le même travail afin de propager l'ornement aux règles déterminées précédemment.

- *QSQR State :*
  Pour chacune des *IDB*, on stocke dans 3 *Map* différentes les *input relation*, *output relation* et les *adorned rules*. Également, on y sauvegarde le nombre total de tuples présents dans les *input relation* et *output relation*.

QSQR Evaluation

qsqrSubroutine

Étant donnée une règle ornée r ayant comme tête un prédicat R^{α}, une *input relation* ($input_R^{\alpha}$), on génère les *QSQTemplate* de la règle qui représente le schéma des relations supplémentaires des atomes. On instancie ensuite des relations sup_i^r fondées sur ce schéma.

L'algorithme suivant a été implémenté :

1. Copy tuples $input_R^{\alpha}$ (that unify with rule head) to sup_0^r
2. For each body atom $A_1, \ldots, A_n$ do:

- If A_i is an *EDB* atom, compute sup_i as projection of $sup^r_{i-1} \bowtie A^I_i$ (where A^I consists of all matches for A in the database)
- If A_i is an *IDB* atom with adorned S^β
  (a) Add new bindings from sup^r_{i-1} to $input^\beta_S$
  (b) If $input^\beta_S$ changed (sup_{i-1}[bound(S^β)] - $input^\beta_S$ is not null), recursively evaluate all rules with head predicate S^β
  (c) Compute sup^r_i as as projection $sup^r_{i-1} \bowtie output^\beta_S$

3. Add tuples in sup^r_n to $output^\alpha_R$

Notons l'utilisation du produit carthésien à la place de la jointure dans le cas particulier des constantes, considérées comme des *EDB*.

NB : Les opérations d'algèbre relationnelle (la jointure $\bowtie$, la soustraction $-$, la projection Π et le produit carthésien $\times$) ont été implémentées dans la classe `Relation`.

QSQR Engine

Étant donné un programme Datalog P et une requête query($\overrightarrow{x}$), on exécute l'algorithme suivant :

1. Create an adorned program P^a
2. Initialize all auxiliary relations to empty sets
3. Evaluate the adorned rule query. Repeat until no new tuples are added to input or output relations
4. Return $output_{query}$

Tests

Validité

Une série de 14 tests a été exécutée pour vérifier la validité de notre programme (*cf.* classe test Java - JUnit - : `src/test/java/fr/univlyon1/mif37/dex/parser/DatalogTest.java`).

Efficacité

Les exemples donnés dans l'archive ont été exécutés avec Datalog ainsi qu'avec le programme implémentant la recherche en top down (*cf.* tableau 1 en Annexe).

- Les temps de calculs de Datalog ont été mesurés avec la commande `/write $computation_time$`.
- Les temps de calculs ont été mesurés sur la même machine.

Bien que supérieurs à ceux mesurés avec Datalog, les temps d'exécution mesurés pour notre programme sont satisfaisants. En effet, une partie importante de la différence constatée peut être expliquée par l'utilisation de classes Java lourdes à l'exécution.

Annexe

Déploiement Maven

Les commandes suivantes sont à exécuter dans le répertoire `query-evaluation`.

- Compilation (necéssaire avant la première exécution et à chaque modification des fichiers sources):

```
$> mvn compile
```

- Exécution :

```
$> mvn exec:java
```

- Exécution des 15 exemples de tests fournis (JUnit 4) :

```
$> mvn test
```

Datalog vs Top Down Query Evaluation

Exemple	Temps DES(ms)	Temps Top Down Evaluation(ms)
1	2	3
2	1	3
3	1	3
4	1	3
5	1	3
6	3	3
7	1	3
8	1	3
9	2	3
10	1	3
11	3	4
12	1	4
13	1	3
14	1	4

Table 1: Temps de calcul par exemple : Datalog vs Top Down Query Evaluation

Diagramme de classe

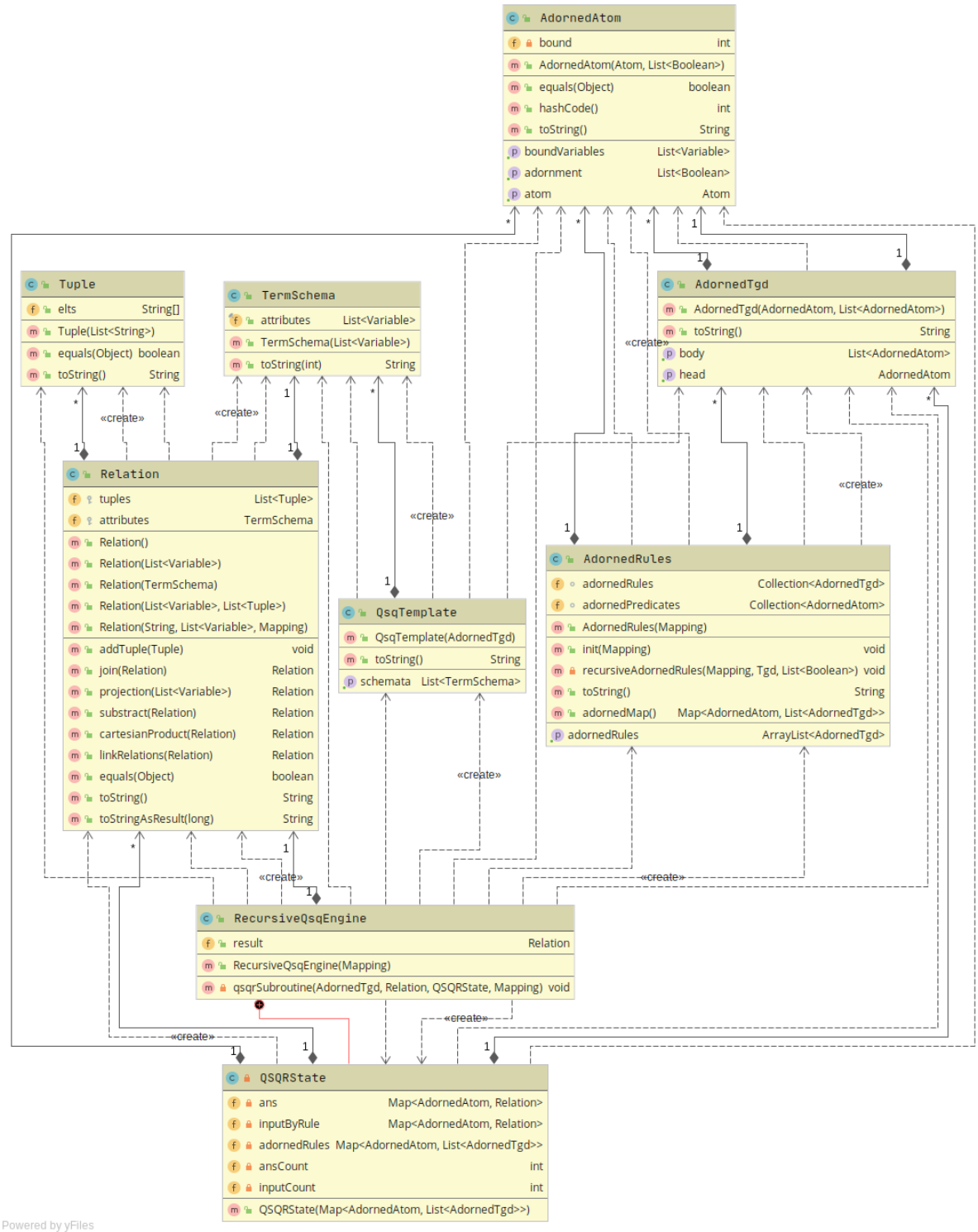

Figure 1: Diagramme de classe