TD3 – Algèbres de collections

UCBL - département informatique - MIF24 BD NoSQL - 2024/2025

Exercice 1 Typage

En supposant les types suivants pour les *builtins* :

- $(>) : int \to int \to bool$
- $(+) : int \to int \to int$
- $sum : [int] \to int$

Donner le type des expressions suivantes :

1. $\lambda r.Map_{\lambda x.(\{A{:}x.A, D{:}x.B+x.C\})}(Filter_{\lambda x.(x.A>x.C)}(r))$
2. $\lambda r.Agg_{\lambda x.(x.A),\texttt{sum},\lambda x.(x.B)}(Filter_{\lambda x(x.C>0)}(r))$
3. $\lambda r.Map_{\lambda x.(\{D{:}x.left.A, E{:}x.right.C\})}(Join_{\lambda y.\lambda z.(y.B=z.B)}(r)(r))$

Exercice 2 Algèbre en SQL

On suppose une relation $R(A, B, C)$ représentée par une collection R de type $< A : int, B : int, C : int >$. Pour chacune des requêtes d'algèbre de collection suivantes, donner un équivalent en SQL.

1. $Map_{\lambda x.(\{A{:}x.A, D{:}x.B+x.C\})}(Filter_{\lambda x.(x.A>x.C)}(R))$
2. $Agg_{\lambda x.(x.A),\texttt{sum},\lambda x.(x.B)}(Filter_{\lambda x(x.C>0)}(R))$
3. $Map_{\lambda x.(\{D{:}x.left.A, E{:}x.right.C\})}(Join_{\lambda y.\lambda z.(y.B=z.B)}(R)(R))$

Exercice 3 SQL en Algèbre

On suppose une relation $R(A, B, C)$ représentée par une collection R de type $< A : int, B : int, C : int >$. Pour chacune des requêtes SQL suivantes, donner un équivalent en algèbre de collections et en MongoDB aggregation pipeline.

1.
```sql
SELECT *
FROM R
WHERE R.A=R.B+R.C
```

2.
```sql
SELECT SUM(A) as D, B, C
FROM R
GROUP BY B, C
ORDER BY B, C
```

3.
```sql
SELECT R.A, R.C, R2.D
FROM R
JOIN (SELECT count(R.A) as D, C
      FROM R
      GROUP BY C) R2
  ON R.C = R2.C
```

4.
```sql
SELECT R1.C, SUM(R1.A)+COUNT(R2.B) as D
FROM R R1 JOIN R R2 ON R1.C = R2.C
GROUP BY R1.C
HAVING R1.C < COUNT(R1.B)
```

Pour la partie MongoDB, on peut se donner une collection de départ pour tester.

```
db.r.insertMany([
    { "A" : 1, "B" : 2, "C" : 3 },
    { "A" : 5, "B" : 2, "C" : 5 },
    { "A" : 5, "B" : 2, "C" : 3 },
    { "A" : 8, "B" : 5, "C" : 3 },
    { "A" : 8, "B" : 5, "C" : 5 },
    { "A" : 1, "B" : 1, "C" : 1 },
    { "A" : 1, "B" : 2, "C" : 1 },
    { "A" : 1, "B" : 1, "C" : 2 },
    { "A" : 1, "B" : 2, "C" : 2 }
])
```

Typage des fonctions

$$\text{(App)}\ \frac{\Gamma \vdash f : \tau \to \tau' \qquad \Gamma \vdash d : \tau}{\Gamma \vdash f(d) : \tau'}$$

$$\text{(Lambda)}\ \frac{\Gamma[x : \tau] \vdash d : \tau'}{\Gamma \vdash \lambda x.d : \tau \to \tau'}$$

$$\text{(Var)}\ \frac{}{\Gamma \vdash x : \tau}\ \text{si}\ x : \tau \in \Gamma$$

Typage des records, des listes

$$\text{(Field)}\ \frac{\Gamma \vdash d :< a : \tau >}{\Gamma \vdash d.a : \tau}$$

$$\text{(Record)}\ \frac{\Gamma \vdash d_1 : \tau_1 \qquad \dots \qquad \Gamma \vdash d_n : \tau_n}{\Gamma \vdash \{a_1 : d_1, \dots, a_n : d_n\} :< a_1 : \tau_1, \dots, a_n : \tau_n >}$$

$$\text{(Singleton)}\ \frac{\Gamma \vdash d : \tau}{\Gamma \vdash [d] : [\tau]}$$

$$\text{(Empty)}\ \frac{}{\Gamma \vdash [] : [\tau]}$$

Types des constantes

$$\text{(Const)}\ \frac{}{\Gamma \vdash c : \tau}$$

en prenant τ et c comme suit :

- type *int* : `1` , `2` , ...
- type *float* : `1.0` , `0.3` , `10.42` , ...
- type *string* : "`truc`" , ...

Sous-typage

$$\text{(Sous-typage)}\ \frac{\Gamma \vdash d : \tau \qquad \tau \preceq \tau'}{\Gamma \vdash d : \tau'}$$

$$\text{(Refl)}\ \frac{}{\tau \preceq \tau}$$

$$\text{(Trans)}\ \frac{\tau \preceq \tau' \qquad \tau' \preceq \tau''}{\tau \preceq \tau''}$$

$$\text{(AddField)}\ \frac{}{< a_1 : \tau_1, \dots, a_k : \tau_k, a_{k+1} : \tau_{k+1} > \preceq < a_1 : \tau_1, \dots, a_k : \tau_k >}$$

$$\text{(STField)}\ \frac{\tau_k \preceq \tau'_k}{< a_1 : \tau_1, \dots, a_k : \tau_k > \preceq < a_1 : \tau_1, \dots, a_k : \tau'_k >}$$

$$\text{(STArray)}\ \frac{\tau \preceq \tau'}{[\tau] \preceq [\tau']} \qquad\qquad \text{(STDict)}\ \frac{\tau \preceq \tau'}{\{\tau\} \preceq \{\tau'\}}$$

FIG. 1 : Typage

$Map_f : [\tau] \to [\tau']$
avec $f : \tau \to \tau'$

$Filter_f : [\tau] \to [\tau]$
avec $f : \tau \to bool$

$Join_f : [\tau_1] \to [\tau_2]$
$\to [< left : \tau_1, right : \tau_2 >]$
avec $f : \tau_1 \to \tau_2 \to bool$

$Agg_{f_m,f_a,k} : [\tau]$
$\to [< key : \tau_k, value : \tau_b >]$
avec $f_m : \tau \to \tau_a$, $f_a : [\tau_a] \to \tau_b$,
$k : \tau \to \tau_k$

$FlatMap_f : [\tau] \to [\tau']$
avec $f : \tau \to [\tau']$

$Sort_f : [\tau] \to [\tau]$
avec $f : \tau \to \tau \to bool$

FIG. 2 : Types des opérateurs de l'algèbre

En MongoDB, on peut appeler le framework d'aggrégation via `db.collection.aggregate(…)`
Voici quelques *stages* utilisables dans le pipeline :

$project spec : ensemble de spécifications de champs. Permet d'ajouter, supprimer ou changer la valeurs de certains champs.

$match spec : conditions sur les champs (c.f. query Mongo). Filtre les éléments du pipeline à la façon d'une requête MongoDB classique.

$lookup spec :

```
{
    from: <collection to join>,
    localField: <field from the input documents>,
    foreignField: <field from the documents of the "from" collection>,
    as: <output array field>
}
```

Effectue une jointure avec la collection spécifiée, mais en combinant chaque docuemnt de la collection courante avec le tableau des documents correspondants.

$unwind spec : chemin de champs avec $ au début. Produit un document pour chaque valeur du tableau contenu dans le champ spécifié en recopiant le reste du document.

$group spec :

```
{
    _id: <expression>, // Group By Expression
    <field1>: { <accumulator1> : <expression1> },
    ...
}
```

Regroupe les documents par valeur décrite dans `_id` et calcule pour chaque champ la valeur aggrégée décrite par l'accumulateur auquel on passe les valeurs calculées par l'expression.

$sort spec : ensemble de champs avec le ordre de tri (1 ou -1)

Réf : https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/

FIG. 3 : Aggrégation pipeline en MongoDB