

Requêtes récursives

Cours L3 Bases de Données

Pierre Senellart



27 février 2019

Motivation : Requêtes récursives

- Langages de requêtes considérés jusqu'à présent (algèbre relationnel, calcul) ont un **horizon limité**
- Certaines structures de données (arbres, graphes) nécessitent des parcours **arbitrairement profonds**
- Applications **naturelles** pour certaines données : listes de discussion, graphes de réseaux de transport, etc.
- **Ce cours** : Comment exprimer des requêtes **récursives** dans les SGBD relationnels ?
- Les SGBDR ne sont **pas toujours** adaptés pour ce type de données/requêtes, cf. SGBD XML ou SGBD graphes
- **Application exemple** : clôture transitive d'un graphe $G(\text{from}, \text{to})$

Plan

Datalog

Récursion et négation

Récursion en SQL

Références

Datalog

- Langage récursif le plus simple : on ajoute de la récursion aux **requêtes conjonctives**
- Inspiré de la **programmation logique**
- Requête (ou **programme**) Datalog : ensemble de règles de productions de **faits intensionnels**
- **Schéma** d'un programme Datalog : schéma relationnel classique (**schéma extensionnel**) + schéma (disjoint) des faits intensionnels (**schéma intensionnel**)
- On fixe une relation distinguée *But* du schéma intensionnel, dont l'arité est l'arité de la requête

Syntaxe

Ensemble fini de règles r de la forme :

$$\underbrace{S(\mathbf{y})}_{\text{tête}} \leftarrow \underbrace{R_1(\mathbf{x}_1), \dots, R_n(\mathbf{x}_n)}_{\text{corps}}$$

avec :

- S relation du schéma **intensionnel**
- R_1, \dots, R_n **relations** du schéma intensionnel ou du schéma extensionnel
- $\mathbf{x}_1, \dots, \mathbf{x}_n, \mathbf{y}$: tuples de **variables** (ou éventuellement de constantes), d'arité compatible avec les relations
- Chaque variable de la tête est **présente** dans le corps

Sémantique par point fixe

- Chaque règle r d'un programme P peut-être vue comme une **requête conjonctive** sur la base de données D :

$$r(D) := \{S(\mathbf{y}) \mid \exists z_1 \dots z_k R_1(\mathbf{x}_1) \in D \wedge \dots \wedge R_n(\mathbf{x}_n) \in D\}$$

où les z_i sont les variables du corps de la règle

- Opérateur de conséquence** Γ_P défini par :

$$\Gamma_P(D) := D \cup \bigcup_{r \in P} \{r(D)\}$$

- On considère la **suite** (D_n) définie par :
 $D_0 = D, D_{n+1} = \Gamma_P(D_n)$
- La sémantique de P sur D est l'ensemble des faits de la relation *But* dans D_∞ , le **point fixe** de la suite (D_n)

Exemple : Clôture transitive

$$But(x, y) \leftarrow G(x, y)$$

$$But(x, y) \leftarrow But(x, z), G(z, y)$$

Détour : Théorème de Knaster–Tarski

Definition

Un **treillis complet** L est un ensemble muni d'un ordre partiel dans lequel **tout sous-ensemble** admet une **borne supérieure** (élément minimum supérieur ou égal à tous les éléments de l'ensemble) et une **borne inférieure**.

Theorem ([Knaster, 1928, Tarski et al., 1955])

*Si L est un **treillis complet** et $f : L \rightarrow L$ une application **croissante** sur L , alors les **points fixes** de f forment un **treillis complet**.*

Application de Knaster–Tarski à Datalog

- L'ensemble des parties d'un ensemble, ordonné par inclusion est toujours un treillis complet ($\bigcup X$ et $\bigcap X$ sont les bornes supérieures et inférieures)
- Γ_P est croissant pour l'inclusion des faits
- Il existe donc un plus petit point fixe de Γ_P , obtenu comme l'intersection de tous les points fixes
- Comme l'ensemble des faits possibles est fini (domaine actif fini), ce plus petit point fixe est obtenu après un nombre fini d'itérations à partir de la base de données initiale
- Définition alternative de l'ensemble des faits produits par un programme Datalog : l'ensemble des faits inclus dans tout modèle du programme Datalog

Arbre de preuve d'un programme Datalog

- Inspiré par la **programmation logique**
- Pour un tuple $But(a_1, \dots, a_n)$, on peut produire une **preuve** de But comme un arbre de preuve :
 - La racine est $But(a_1, \dots, a_n)$
 - Tout noeud interne est l'instantiation de la tête d'une règle, ses enfants une instantiation compatible du corps de cette même règle
 - Les feuilles sont des faits extensionnels
- Utile pour **raisonner** sur les programmes Datalog

Datalog non récursif

Definition

Le graphe d'un programme Datalog est le graphe dont les noeuds sont les relations du schéma intensionnel et dans lequel il existe une arête (R, S) s'il existe une règle avec R dans le corps et S dans la tête. Un programme Datalog est **non récursif** si ce graphe est **acyclique**.

- Même pouvoir d'expression que les **unions de requêtes conjonctives**
- Mais exponentiellement plus **compact** !

Plan

Datalog

Récursion et négation

While en algèbre relationnelle

Logique à point fixe

Datalog à négation

Récursion en SQL

Références

Plan

Datalog

Récursion et négation

While en algèbre relationnelle

Logique à point fixe

Datalog à négation

Récursion en SQL

Références

Opérateur While

- Algèbre : essentiellement **programmation impérative** (contrairement au calcul)
- On ajoute à l'algèbre :
 - la possibilité de définir des **variables intermédiaires** et de leur **affecter** une valeur (ne change pas le pouvoir d'expression)
 - un **opérateur while** de la forme :

while change do

...affectation de valeur à une ou plusieurs variables...

done

- **Sémantique** : le contenu de la boucle est exécuté **tant que** les affectations changent les variables sous-jacentes
- **Boucles infinies** possibles !

While inflationniste vs non-inflationniste

- Deux variantes :

Non-inflationniste Opérateur d'affectation $:=$, affectation arbitraire

Inflationniste Opérateur d'affectation $+=$, la variable affectée ne peut que **croître**

- Boucles infinies **impossibles avec un `while` inflationniste** s'il n'y a pas d'arithmétique (parce que le domaine actif est fini)
- Variante non-inflationniste **plus expressive**

Exemple : Clôture transitive

On introduit un schéma de relation C avec attributs *from* et *to*, tout comme G .

Non-inflationniste

$C := G$

while change do

$C := C \cup \pi_{from,to}(\rho_{to \rightarrow int}(C) \bowtie \rho_{from \rightarrow int}(G))$

done

C

Inflationniste

$C += G$

while change do

$C += \pi_{from,to}(\rho_{to \rightarrow int}(C) \bowtie \rho_{from \rightarrow int}(G))$

done

C

Plan

Datalog

Récursion et négation

While en algèbre relationnelle

Logique à point fixe

Datalog à négation

Récursion en SQL

Références

Point fixe non-inflationniste

- On ajoute au calcul relationnel une construction de **point fixe**
- Supposons $\varphi(T)$ formule du calcul, mentionnant les relations du schéma ainsi qu'une **nouvelle relation T** , et ayant pour variables libres x_1, \dots, x_n
- Alors $\mu_T[\varphi(T)](x_1, \dots, x_n)$ est une formule du **calcul avec point-fixe**
- **Sémantique** : On considère le **plus petit point fixe** de la relation T en remplaçant T à chaque étape par l'ensemble des faits de la forme $T(x_1, \dots, x_n)$ pour $\varphi(T)(x_1, \dots, x_n)$ satisfait (en partant de $T = \emptyset$)
- **Équivalent** à l'algèbre avec **while** non-inflationniste !

Point fixe inflationniste

- On ajoute au calcul relationnel une construction de **point fixe**
- Supposons $\varphi(T)$ formule du calcul, mentionnant les relations du schéma ainsi qu'une **nouvelle relation T** , et ayant pour variables libres x_1, \dots, x_n
- Alors $\mu_T^+[\varphi(T)](x_1, \dots, x_n)$ est une formule du **calcul avec point-fixe**
- **Sémantique** : On considère le **plus petit point fixe** de la relation T en ajoutant à T à chaque étape l'ensemble des faits de la forme $T(x_1, \dots, x_n)$ pour $\varphi(T)(x_1, \dots, x_n)$ satisfait (en partant de $T = \emptyset$)
- **Équivalent** à l'algèbre avec **while** inflationniste !

Exemple : Clôture transitive

Non-inflationniste

$$\{(x, y) \mid \mu_C [G(x, y) \vee C(x, y) \vee (\exists z C(x, z) \wedge G(z, y))](x, y)\}$$

Inflationniste

$$\{(x, y) \mid \mu_C^+ [G(x, y) \vee (\exists z C(x, z) \wedge G(z, y))](x, y)\}$$

Logique du second ordre (SO)

- Extension de la logique du premier ordre dans laquelle il est possible d'utiliser des **quantificateurs sur des relations**
- Par exemple, on peut exprimer en logique du second ordre qu'un graphe G est **biparti** :

$$\begin{aligned} \exists X \exists Y \forall x \forall y [G(x, y) \rightarrow ((X(x) \wedge Y(y)) \vee (X(y) \wedge Y(x)))] \\ \wedge \neg(\exists x X(x) \wedge Y(x)) \end{aligned}$$

- Peut être aussi vu comme un **langage de requêtes**, plus puissant que le calcul relationnel

Point fixe inflationniste et SO

- Dans certains cas, **traduction naturelle** de logique à point fixe vers logique du second-ordre
- En particulier, pour un point fixe inflationniste :

$$\{(x_1, \dots, x_n) \mid \mu_T^+[\varphi(T)](x_1, \dots, x_n)\}$$

s'exprime **en SO** :

$$\{(x_1, \dots, x_n) \mid \forall T(\forall y_1 \dots \forall y_n \varphi(T)(y_1, \dots, y_n) \rightarrow T(y_1, \dots, y_n)) \rightarrow T(x_1, \dots, x_n)\}$$

- Application immédiate de **Knaster–Tarski** ! Le plus petit point fixe (calculé par μ^+) de l'opérateur qui ajoute à T les tuples tel que $\varphi(T)$ est satisfait est croissant, et donc l'intersection de tous ($\forall T$) les modèles de φ

Exemple : Clôture transitive

$$\{ (x, y) \mid$$

$$\forall C [\forall f \forall t (G(f, t) \vee (\exists z C(f, z) \wedge G(z, t))) \rightarrow C(f, t)]$$

$$\rightarrow C(x, y) \}$$

Plan

Datalog

Récursion et négation

While en algèbre relationnelle

Logique à point fixe

Datalog à négation

Récursion en SQL

Références

Datalog avec négation dans le corps

- On ajoute à Datalog la possibilité d'écrire $\neg R(x_1, \dots, x_n)$ dans le **corps** d'une règle
- R peut être **extensionnel** ou **intensionnel**
- **Sémantique domaine actif** (nombre fini de faits tel que $\neg R(x_1, \dots, x_n)$)
- Définition par **point fixe** :

$$r : S(y) \leftarrow R_1(\mathbf{x}_1), \dots, \neg R_i(\mathbf{x}_i), \dots, R_n(\mathbf{x}_n)$$
$$r(D) := \{S(y) \mid \exists z_1 \dots z_k R_1(\mathbf{x}_1) \in D \wedge R_i(\mathbf{x}_i) \notin D$$
$$\wedge \dots \wedge R_n(\mathbf{x}_n) \in D\}$$

- L'opérateur de conséquence reste **croissant**, terminaison assurée
- Équivalent au calcul avec point fixe **inflationniste**, à l'algèbre avec **while inflationniste**

Exemple : Clôture transitive du complément

$$But(x, y) \leftarrow \neg G(x, y)$$

$$But(x, y) \leftarrow But(x, z), \neg G(z, y)$$

Plan

Datalog

Récursion et négation

Récursion en SQL

Références

Common Table Expressions (CTE)

- Seule possibilité en SQL d'introduire de la récursion (on parle aussi de **requêtes hiérarchiques**)
- **Idée** : On nomme une table, dont la définition s'utilise elle-même
- Inspiré des logiques à **point fixe inflationniste**, mais aussi des fonctions récursives dans les langages de programmation (penser au **let rec** de Caml)

Syntaxe

```

WITH RECURSIVE T(x1, ..., xn) AS (
  SELECT -- cas de base
UNION
  SELECT -- cas récursif, utilisant la table T
)
SELECT * FROM T;

```

Cette forme précise, utilisant **UNION** (ou **UNION ALL**) entre cas de base et cas récursif est **obligatoire**

Sémantique

- On évalue la requête **itérativement**, en partant de $T = \emptyset$ et en remplaçant à chaque étape T par le résultat de l'évaluation de la définition de T
- On s'arrête quand un **point fixe est atteint**
- La requête définissant T doit être **monotone** (on ne peut à chaque étape que faire croître T)
- Optimisations possibles (et appliquées) permettant de ne tenir compte que des **nouveaux faits** à chaque étape (caractère inflationniste)
- Boucle infinie **possible** si on crée de nouvelles valeurs

Exemple : Clôture transitive

```
WITH RECURSIVE C(f,t) AS (  
  SELECT * FROM G  
  UNION  
  SELECT C.f, G.t FROM C JOIN G ON C.t=G.f  
)  
SELECT * FROM C;
```

Notes sur le support historique

- CTE normalisées **tardivement** [ISO, 1999] en SQL
- Extensions historiques **non compatibles** des SGBD (**CONNECT BY** introduit par Oracle, reprises dans certains autres SGBD)
- Support maintenant **correct**, mais uniquement dans les SGBD relativement récents
- Les requêtes récursives ne sont pas une priorité des SGBD, optimisations à la traîne (dans PostgreSQL, les CTE sont des **barrières d'optimisation**)

Plan

Datalog

Récursion et négation

Récursion en SQL

Références

Références

- Datalog : chapitre 12 de [Abiteboul et al., 1995]
- While, Logiques à point fixe, Datalog à négation : chapitre 14 de [Abiteboul et al., 1995]
- Récursion en PostgreSQL : <https://www.postgresql.org/docs/current/static/queries-with.html>

Bibliographie I

Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

ISO. *ISO 9075:1999: SQL*. International Standards Organization, 1999.

Bronisław Knaster. Un théoreme sur les fonctions d'ensembles. *Ann. Soc. Polon. Math*, 6(133) :2013134, 1928.

Alfred Tarski et al. A lattice-theoretical fixpoint theorem and its applications. *Pacific journal of Mathematics*, 5(2) : 285–309, 1955.