Vues, provenance, déclencheurs

Cours L3 Bases de Données

Pierre Senellart

26 avril 2017

Plan

Vues

- Les vues sont des requêtes auxquelles l'on donne un nom
- Elles s'utilisent comme des tables dans d'autres requêtes
- Sémantique : on remplace la vue par le résultat de l'évaluation de la requête correspondante

En SQL

```
CREATE VIEW vue AS SELECT ...
```

```
CREATE MATERIALIZED VIEW vue AS SELECT ...
```

Vues virtuelles et vues matérialisées

- Une vue peut être virtuelle (par défaut) ou matérialisée
- Pas de différence de sémantique
- Différence opérationnelle, avec impact sur l'efficacité de l'évaluation de requêtes :

  vue virtuelle : la requête définissant la vue est réévaluée à chaque fois que la vue est utilisée dans une requête

  vue matérialisée : la requête définissant la vue est évaluée à la création de la vue et le résultat est stocké dans une table auxiliaire ; cette table est directement utilisée à chaque fois que la vue est utilisée dans une requête

Usage des vues

Indépendance logique : une application peut accéder à des vues, sans avoir besoin de connaître l'organisation effective des données dans la base (qui peut changer de manière transparente, en redéfinissant les vues)

Contrôle d'accès : des droits d'accès différents peuvent être donnés aux tables de base et à des vues, pour qu'un utilisateur ou application donnée n'ait accès qu'à un ensemble restreint du contenu de la base

Intégration de données : des vues peuvent être définies pour regrouper les données de plusieurs sources ayant des schémas relationnels différents

Optimisation : des vues matérialisées peuvent être définies pour des requêtes fréquentes, pour éviter d'avoir à les ré-évaluer à chaque fois

Répondre à des requêtes grâce à des vues

- Problème : étant donné une requête sur les tables de base et un ensemble de vues, calculer une réécriture équivalente à la requête utilisant uniquement les vues
- Parfois impossible de trouver une réécriture équivalente, mais possible de trouver une réécriture maximalement contenue
- Très important en intégration de données, quand les vues sont utilisées pour décrire les données accessibles dans les différentes sources en fonction d'un schéma global
- Parfois nécessaire de tenir compte du coût d'accès à chacune des vues
- Très étudié, nombreux algorithmes proposés pour différents langages de requêtes et de vue, voir la vue d'ensemble du sujet [Halevy, 2001]

Vues et mises à jour

Les vues interagissent de manière complexe avec les mises à jour (insertion, modification, suppression).

Maintenance de vues : quand une mise à jour a lieu sur les tables de base, cette mise à jour doit être reflétée dans les vues

- Pas de souci pour les vues virtuelles
- Plus compliqué pour les vues matérialisées qui doivent être maintenues en fonction des mises à jour

Vues et mises à jour

Les vues interagissent de manière complexe avec les mises à jour (insertion, modification, suppression).

Maintenance de vues : quand une mise à jour a lieu sur les tables de base, cette mise à jour doit être reflétée dans les vues

- Pas de souci pour les vues virtuelles
- Plus compliqué pour les vues matérialisées qui doivent être maintenues en fonction des mises à jour

Mise à jour au travers de vues : on veut pouvoir dans certaines circonstances faire une opération de mise à jour sur une vue, et que cela résulte en des mises à jour appropriées des tables de base

Vues et mises à jour

Les vues interagissent de manière complexe avec les mises à jour (insertion, modification, suppression).

Maintenance de vues : quand une mise à jour a lieu sur les tables de base, cette mise à jour doit être reflétée dans les vues

- Pas de souci pour les vues virtuelles
- Plus compliqué pour les vues matérialisées qui doivent être maintenues en fonction des mises à jour

Mise à jour au travers de vues : on veut pouvoir dans certaines circonstances faire une opération de mise à jour sur une vue, et que cela résulte en des mises à jour appropriées des tables de base

Comment faire ça ?

Plan

Modèle de données

- Modèle de données relationnel : données décomposées en relations, avec attributs étiquetés...

Modèle de données

- Modèle de données relationnel : données décomposées en relations, avec attributs étiquetés...

nom	poste	ville
John	Directeur	New York
Paul	Gardien	New York
Dave	Analyste	Paris
Ellen	Agent	Berlin
Magdalen	Agent double	Paris
Nancy	DRH	Paris
Susan	Analyste	Berlin

Modèle de données

- Modèle de données relationnel : données décomposées en relations, avec attributs étiquetés...
- ... et une annotation de provenance supplémentaire pour chaque tuple (la voir comme un identifiant de tuple)

nom	poste	ville	prov
John	Directeur	New York	t_1
Paul	Gardien	New York	t_2
Dave	Analyste	Paris	t_3
Ellen	Agent	Berlin	t_4
Magdalen	Agent double	Paris	t_5
Nancy	DRH	Paris	t_6
Susan	Analyste	Berlin	t_7

Provenance booléenne [Imielinski and Jr., 1984]

- $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ ensemble fini d'événements booléens
- annotation de provenance : fonction booléenne sur $\mathcal{X}$, c.-à-d., une fonction de la forme : $(\mathcal{X} \to \{\bot, \top\}) \to \{\bot, \top\}$
- Interprétation : sémantique de mondes possibles
  - chaque valuation $\nu : \mathcal{X} \to \{\bot, \top\}$ désigne un monde possible de la base de données
  - la provenance d'un tuple ν évalue à $\bot$ ou $\top$ suivant si ce tuple existe dans ce monde possible
  - par exemple, si chaque tuple de la base est annoté avec la fonction indicatrice d'un événement booléen distinct, l'ensemble des mondes possibles est l'ensemble de toutes les sous-instances

Exemples de mondes possibles

nom	poste	ville	prov
John	Directeur	New York	t_1
Paul	Gardien	New York	t_2
Dave	Analyste	Paris	t_3
Ellen	Agent	Berlin	t_4
Magdalen	Agent double	Paris	t_5
Nancy	DRH	Paris	t_6
Susan	Analyste	Berlin	t_7

$$\nu : \begin{array}{ccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 \\ \top & \top & \top & \top & \top & \top & \top \end{array}$$

Exemples de mondes possibles

nom	poste	ville	prov
John	Directeur	New York	t_1
Dave	Analyste	Paris	t_3
Magdalen	Agent double	Paris	t_5
Susan	Analyste	Berlin	t_7

$$\nu : \begin{array}{ccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 \\ \top & \bot & \top & \bot & \top & \bot & \top \end{array}$$

Provenance booléenne des résultats de requêtes

- $\nu(D)$: la sous-instance de D où tous les tuples dont l'annotation de provenance évalue à $\bot$ par ν sont enlevés
- La provenance booléenne $\text{prov}_{q,D}(t)$ d'un tuple $t \in q(D)$ est la fonction :

$$\nu \mapsto \begin{cases} \top \text{ si } t \in q(\nu(D)) \\ \bot \text{ sinon} \end{cases}$$

Exemple (Quelles villes sont dans la table Personnel ?)

nom	poste	ville	prov
John	Directeur	New York	t_1
Paul	Gardien	New York	t_2
Dave	Analyste	Paris	t_3
Ellen	Agent	Berlin	t_4
Magdalen	Agent double	Paris	t_5
Nancy	DRH	Paris	t_6
Susan	Analyste	Berlin	t_7

ville	prov
New York	$t_1 \vee t_2$
Paris	$t_3 \vee t_5 \vee t_6$
Berlin	$t_4 \vee t_7$

Semi-anneau commutatif $(K, \mathbb{0}, \mathbb{1}, \oplus, \otimes)$

- Ensemble K avec éléments distingués $\mathbb{0}$, $\mathbb{1}$
- $\oplus$ opérateur associatif, commutatif, avec neutre $\mathbb{0}_K$:
  - $a \oplus (b \oplus c) = (a \oplus b) \oplus c$
  - $a \oplus b = b \oplus a$
  - $a \oplus \mathbb{0} = \mathbb{0} \oplus a = a$
- $\otimes$ opérateur associatif, commutatif, avec neutre $\mathbb{1}_K$:
  - $a \otimes (b \otimes c) = (a \otimes b) \otimes c$
  - $a \otimes b = b \otimes a$
  - $a \otimes \mathbb{1} = \mathbb{1} \otimes a = a$
- $\otimes$ distribue sur $\oplus$:

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$$

- $\mathbb{0}$ est annulateur pour $\otimes$:

$$a \otimes \mathbb{0} = \mathbb{0} \otimes a = \mathbb{0}$$

Exemples de semi-anneaux

- $(\{\text{fonctions booléennes sur } \mathcal{X}\}, \bot, \top, \vee, \wedge)$: semi-anneau des fonctions booléennes sur $\mathcal{X}$
- $(\mathcal{P}(\mathcal{P}(\mathcal{X})), \emptyset, \{\emptyset\}, \cup, \Cup)$: Semi-anneau Why sur $\mathcal{X}$
  $(A \Cup B := \{a \cup b \mid a \in A, b \in B\})$

Provenance de semi-anneau [Green et al., 2007]

- On fixe un semi-anneau $(K, \mathbb{0}, \mathbb{1}, \oplus, \otimes)$
- On suppose que les annotations de provenance sont dans K
- On considère une requête q de l'algèbre relationnelle positive (sélection, projection, renommage, produit cartésien, union ; jointures simulables avec renommage, produit cartésien, sélection, projection)
- On définit la sémantique de la provenance d'un tuple $t \in q(D)$ récursivement sur la structure de q

Sélection, renommage

Les annotations de provenance des tuples sélectionnés sont inchangées

Exemple $(\rho_{\text{nom}\to\text{n}}(\sigma_{\text{ville}=\text{“New York”}}(R)))$

nom	poste	ville	prov
John	Directeur	New York	t_1
Paul	Gardien	New York	t_2
Dave	Analyste	Paris	t_3
Ellen	Agent	Berlin	t_4
Magdalen	Agent double	Paris	t_5
Nancy	DRH	Paris	t_6
Susan	Analyste	Berlin	t_7

n	poste	ville	prov
John	Directeur	New York	t_1
Paul	Gardien	New York	t_2

Projection

Les annotations de provenance des tuples identiques, fusionnés, sont $\oplus$-ées

Exemple ($\pi_{\text{ville}}(R)$)

nom	poste	ville	prov
John	Directeur	New York	t_1
Paul	Gardien	New York	t_2
Dave	Analyste	Paris	t_3
Ellen	Agent	Berlin	t_4
Magdalen	Agent double	Paris	t_5
Nancy	DRH	Paris	t_6
Susan	Analyste	Berlin	t_7

ville	prov
New York	$t_1 \oplus t_2$
Paris	$t_3 \oplus t_5 \oplus t_6$
Berlin	$t_4 \oplus t_7$

Union

Les annotations de provenance des tuples identiques, fusionnés, sont $\oplus$-ées

Exemple

$\pi_{\text{ville}}(\sigma_{\textit{starts-with}(\text{poste},\text{“Agent”})}(R)) \cup \pi_{\text{ville}}(\sigma_{\text{poste}=\text{“Analyste”}}(R))$

nom	poste	ville	prov
John	Directeur	New York	t_1
Paul	Gardien	New York	t_2
Dave	Analyste	Paris	t_3
Ellen	Agent	Berlin	t_4
Magdalen	Agent double	Paris	t_5
Nancy	DRH	Paris	t_6
Susan	Analyste	Berlin	t_7

ville	prov
Paris	$t_3 \oplus t_5$
Berlin	$t_4 \oplus t_7$

Produit cartésien

Les annotations de provenance des tuples combinés sont $\otimes$-ées

Exemple

$\pi_{\text{ville}}(\sigma_{\textit{starts-with}(\text{poste},\text{“Agent”})}(R)) \bowtie \pi_{\text{ville}}(\sigma_{\text{poste}=\text{“Analyste”}}(R))$

nom	poste	ville	prov
John	Directeur	New York	t_1
Paul	Gardien	New York	t_2
Dave	Analyste	Paris	t_3
Ellen	Agent	Berlin	t_4
Magdalen	Agent double	Paris	t_5
Nancy	DRH	Paris	t_6
Susan	Analyste	Berlin	t_7

ville	prov
Paris	$t_3 \otimes t_5$
Berlin	$t_4 \otimes t_7$

Que peut-on faire avec ça ?

Fonctions booléennes : provenance booléenne, comme définie précédemment

Semi-anneau Why : Why-provenance [Buneman et al., 2001], ensemble des combinaisons de tuples nécessaires à l'existence d'un tuple

. . . et plein d'autres choses en choisissant d'autres semi-anneaux !

Exemple de Why-provenance

$$\pi_{\text{ville}}(\sigma_{\text{nom}<\text{nom2}}(\pi_{\text{nom,ville}}(R) \bowtie \rho_{\text{nom}\to\text{nom2}}(\pi_{\text{nom,ville}}(R))))$$

nom	poste	ville	prov
John	Directeur	New York	t_1
Paul	Gardien	New York	t_2
Dave	Analyste	Paris	t_3
Ellen	Agent	Berlin	t_4
Magdalen	Agent double	Paris	t_5
Nancy	DRH	Paris	t_6
Susan	Analyste	Berlin	t_7

ville	prov
New York	$\{\{t_1, t_2\}\}$
Paris	$\{\{t_3, t_5\}, \{t_3, t_6\}, \{t_5, t_6\}\}$
Berlin	$\{\{t_4, t_7\}\}$

Remarques [Green et al., 2007]

- Le calcul de la provenance a un surcoût polynomial en temps
- Deux requêtes équivalentes peuvent avoir deux annotations différentes de provenance sur la même base de données, dans certains semi-anneaux (par exemple, dans le semi-anneau Why)

Plan

Insertions

Pour l'algèbre relationnelle positive, possible de calculer l'impact d'une insertion de manière incrémentale :

$$\sigma_\varphi(R \cup \Delta R) = \sigma_\varphi(R) \cup \sigma_\varphi(\Delta R)$$
$$\Pi_X(R \cup \Delta R) = \Pi_X(R) \cup \Pi_X(\Delta R)$$
$$\rho_{A\to B}(R \cup \Delta R) = \rho_{A\to B}(R) \cup \rho_{A\to B}(\Delta R)$$
$$(R \cup \Delta R) \cup (S \cup \Delta S) = (R \cup S) \cup \Delta R \cup \Delta S$$
$$(R \cup \Delta R) \times (S \cup \Delta S) = (R \times S) \cup (R \times \Delta S) \cup (\Delta R \times S) \cup (\Delta R \times \Delta S)$$
$$(R \cup \Delta R) \bowtie (S \cup \Delta S) = (R \bowtie S) \cup (R \bowtie \Delta S) \cup (\Delta R \bowtie S) \cup (\Delta R \bowtie \Delta S)$$

Suppressions

- Il suffit d'utiliser la provenance booléenne !
- Enlever tous les tuples dont l'annotation de provenance s'évalue à $\bot$

Suppressions

- Il suffit d'utiliser la provenance booléenne !
- Enlever tous les tuples dont l'annotation de provenance s'évalue à $\bot$

nom	poste	ville	prov
John	Directeur	New York	t_1
Paul	Gardien	New York	t_2
Dave	Analyste	Paris	t_3
Ellen	Agent	Berlin	t_4
Magdalen	Agent double	Paris	t_5
Nancy	DRH	Paris	t_6
Susan	Analyste	Berlin	t_7

ville	prov
New York	$t_1 \wedge t_2$
Paris	$(t_3 \wedge t_5) \vee (t_3 \wedge t_6) \vee (t_5 \wedge t_6)$
Berlin	$t_4 \wedge t_7$

Si t_1 disparaît

Suppressions

- Il suffit d'utiliser la provenance booléenne !
- Enlever tous les tuples dont l'annotation de provenance s'évalue à $\bot$

nom	poste	ville	prov
John	Directeur	New York	t_1
Paul	Gardien	New York	t_2
Dave	Analyste	Paris	t_3
Ellen	Agent	Berlin	t_4
Magdalen	Agent double	Paris	t_5
Nancy	DRH	Paris	t_6
Susan	Analyste	Berlin	t_7

ville	prov
New York	$t_1 \wedge t_2$
Paris	$(t_3 \wedge t_5) \vee (t_3 \wedge t_6) \vee (t_5 \wedge t_6)$
Berlin	$t_4 \wedge t_7$

Si t_1 disparaît, New York disparaît du résultat de la vue.

Et dans PostgreSQL ?

- La maintenance automatique de vue n'est pas implémentée !
- On peut maintenir manuellement une vue avec :

  `REFRESH MATERIALIZED VIEW vue`
- Pour une maintenance automatique, transformer la vue matérialisée en une vraie table, et ajouter des déclencheurs aux tables de base pour traiter les opérations de mise à jour de manière ad hoc

```
CREATE TRIGGER tab_trigger
  AFTER INSERT ON tab
  FOR EACH ROW EXECUTE PROCEDURE tab_insert()
```

Plan

Insertions

Difficile en général :

- En cas de jointure avec valeur de jointure projetée, il faut « inventer » une valeur de jointure pour ajouter des tuples avec cette valeur dans plusieurs tables de base
- En cas de projection simple, peupler la table de base de valeurs **NULL**
- En cas d'agrégation : essentiellement impossible sauf à définir une stratégie adaptée à chaque problème

Suppressions [Buneman et al., 2002]

- Cas d'utilisation de la Why-provenance !
- Pour supprimer un tuple t dans le résultat d'une vue, sélectionner un sous-ensemble de tuples minimal (en taille, ou en termes d'effet de bord sur les autres tuples de la vue supprimés) dont l'annotation est présente dans chaque ensemble d'annotations de la Why-provenance de t
- NP-complet en général

Suppressions [Buneman et al., 2002]

- Cas d'utilisation de la Why-provenance !
- Pour supprimer un tuple t dans le résultat d'une vue, sélectionner un sous-ensemble de tuples minimal (en taille, ou en termes d'effet de bord sur les autres tuples de la vue supprimés) dont l'annotation est présente dans chaque ensemble d'annotations de la Why-provenance de t
- NP-complet en général

nom	poste	ville	prov
John	Directeur	New York	t_1
Paul	Gardien	New York	t_2
Dave	Analyste	Paris	t_3
Ellen	Agent	Berlin	t_4
Magdalen	Agent double	Paris	t_5
Nancy	DRH	Paris	t_6
Susan	Analyste	Berlin	t_7

ville	prov
New York	$\{\{t_1, t_2\}\}$
Paris	$\{\{t_3, t_5\}, \{t_3, t_6\}, \{t_5, t_6\}\}$
Berlin	$\{\{t_4, t_7\}\}$

Pour supprimer Paris

Suppressions [Buneman et al., 2002]

- Cas d'utilisation de la Why-provenance !
- Pour supprimer un tuple t dans le résultat d'une vue, sélectionner un sous-ensemble de tuples minimal (en taille, ou en termes d'effet de bord sur les autres tuples de la vue supprimés) dont l'annotation est présente dans chaque ensemble d'annotations de la Why-provenance de t
- NP-complet en général

nom	poste	ville	prov
John	Directeur	New York	t_1
Paul	Gardien	New York	t_2
Dave	Analyste	Paris	t_3
Ellen	Agent	Berlin	t_4
Magdalen	Agent double	Paris	t_5
Nancy	DRH	Paris	t_6
Susan	Analyste	Berlin	t_7

ville	prov
New York	$\{\{t_1, t_2\}\}$
Paris	$\{\{t_3, t_5\}, \{t_3, t_6\}, \{t_5, t_6\}\}$
Berlin	$\{\{t_4, t_7\}\}$

Pour supprimer Paris, supprimer deux tuples parmi t_3, t_5, t_6.

Et dans PostgreSQL ?

- Le standard SQL n'autorise les mises à jour qu'à travers des vues très simples (sélection, projection, renommage) :
  - pas de jointure
  - pas d'union, d'intersection, de différence
  - pas d'agrégation
  - pas de requête récursive
  - pas de sémantique ensembliste
- Pour les autres cas, possible de définir un déclencheur sur la vue

```
CREATE TRIGGER vue_trigger
  INSTEAD OF INSERT OR UPDATE OR DELETE ON vue
  FOR EACH ROW EXECUTE PROCEDURE vue_maj()
```

Plan

Déclencheurs (triggers) en PostgreSQL

- Fonctions définissables dans le SGBD, se déclenchant quand certains événements se produisent :
  - avant une mise à jour sur une table ou une vue **BEFORE**
  - après une mise à jour sur une table ou une vue **AFTER**
  - à la place d'une mise à jour sur une vue **INSTEAD OF**
- Peuvent être déclenchés une fois par tuple affecté (**FOR EACH ROW**) ou une fois par opération (**FOR EACH STATEMENT**)
- Peuvent être définies dans plusieurs langages de programmation, le plus courant étant PL/pgSQL
- cf. `https://www.postgresql.org/docs/9.6/static/sql-createtrigger.html`

PL/pgSQL

- Langage de programmation impératif de PostgreSQL
- cf. `https://www.postgresql.org/docs/9.6/static/plpgsql.html`
- Permet de définir des déclencheurs :

```
CREATE FUNCTION tab_insert() RETURNS TRIGGER
  LANGUAGE plpgsql
AS $$
  BEGIN
    ...
    RETURN NEW;
  END;
$$;
```

- Au sein d'une fonction définissant un déclencheur ligne par ligne, **OLD** est l'ancienne valeur du tuple (pour modification ou suppression), **NEW** est la nouvelle valeur (pour insertion ou modification)

Plan

Références

- L'article présentant les semi-anneaux de provenance [Green et al., 2007]
- La vue d'ensemble de réponse à des requêtes en utilisant des vues [Halevy, 2001]
- Chapitre 22 de [Abiteboul et al., 1995]
- La documentation de PostgreSQL pour les déclencheurs et PL/pgSQL `https://www.postgresql.org/docs/9.6/`

Bibliographie I

Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. ISBN 0-201-53771-0. URL `http://www-cse.ucsd.edu/users/vianu/book.html`.

Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where : A characterization of data provenance. In *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings.*, 2001.

Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. On propagation of deletions and annotations through views. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 150–158, 2002. doi : 10.1145/543613.543633. URL `http://doi.acm.org/10.1145/543613.543633`.

Bibliographie II

Todd J Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, 2007.

Alon Y. Halevy. Answering queries using views : A survey. *VLDB J.*, 10(4) :270–294, 2001. doi : 10.1007/s007780100054. URL `http://dx.doi.org/10.1007/s007780100054`.

Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.