

Interrogation des données : compléments

BD-BioInfo Bases de données

Cristina Sirangelo
IRIF, Université Paris Diderot
cristina@irif.fr

Jointures externes

Jointures externes (OUTER JOIN)

- L'opérateur JOIN (simple ou NATURAL) peut être modifié avec la modalité OUTER
LEFT [OUTER] JOIN
RIGHT [OUTER] JOIN
FULL [OUTER] JOIN
(OUTER optionnel)
- Effet :
 - ▶ d'abord la jointure est effectuée normalement,
 - ▶ des lignes additionnelles sont ajoutées au résultat (voir transparents suivants)

Jointure externe gauche (LEFT [OUTER] JOIN)

TX

A	B
1	4
2	3
2	8
4	6

TY

B	C
7	7
3	8
5	6
4	2
3	11

`select * from TX NATURAL LEFT JOIN TY`



A	B	C
1	4	2
2	3	8
2	8	null
4	6	null
2	3	11

Chaque ligne de TX (la table de gauche) doit appartenir à la réponse, les lignes sans compagnon sont complétées par les valeurs null.

Autres types de jointure externe

TX RIGHT JOIN TY — en plus de TX join TY
on garde **toutes les lignes de TY** sans compagnon dans TX,
complétées par des NULL

TX FULL JOIN TY — en plus de TX join TY
on garde **toutes les lignes de TX et TY** qui n'ont
pas de compagnon dans l'autre table,
complétées par des NULL

Resumé : les opérateurs de jointure

T1 join T2 on join_condition

T1 left join T2 on join_condition

T1 right join T2 on join_condition

T1 full join T2 on join_condition

T1 natural join T2

T1 natural left join T2

T1 natural right join T2

T1 natural full join T2

T1 cross join T2 (la même chose que T1,T2)

Exemples d'utilisation des jointures externes

Pour ne pas perdre de lignes dans le résultat

Trouver les titres de tous livres de la bibliothèque avec les identifiants de leurs exemplaires, quand il en existe :

```
select titre, ide  
from livre natural left join exemplaire
```

Les livres qui sont catalogués mais n'ont pas d'exemplaire seront renvoyés sous la forme

titre	ide
"Les misérables"	NULL

Exemples d'utilisation des jointures externes

Pour remplacer la negation :

Trouver les titres de livres **anonymes** (c'est-à-dire qui n'ont pas d'auteur) :

```
select titre  
from livre natural left join livre_auteur  
where id_auteur is null
```

Remarquer que on n'a pas besoin de la jointure externe pour trouver les titres de livres qui ont un auteur :

```
select titre from livre natural join livre_auteur ;
```


Exemples d'utilisation des jointures externes

Pour remplacer la negation :

Trouver les identifiants des exemplaires **disponibles** (i.e. sans emprunt en cours):

```
select ide
from exemplaire Ex LEFT JOIN emprunt E
      ON ( Ex.ide = E.ide AND date_effective is NULL)
where date_empr is null
```

Exemples d'utilisation des jointures externes

Pour remplacer la negation :

Trouver le nombre (d'exemplaires) de livres que détient chaque lecteur en ce moment, inclure les lecteurs avec 0 emprunts en cours:

```
(select nom, prenom, count(*) as nb_livres
from lecteur natural join emprunt
where date_effective is null
group by id_lecteur, nom, prenom)
union
(select nom, prenom, 0
from lecteur L left join emprunt E
ON (L.id_lecteur = E.id_lecteur
and date_effective is null)
where id_e is null);
```

Exemples d'utilisation des jointures externes

Pour remplacer la negation :

Trouver le nombre (d'exemplaires) de livres que détient chaque lecteur en ce moment, inclure les lecteurs avec 0 emprunts en cours:

```
(select nom, prenom, count(*) as nb_livres  
from lecteur natural join emprunt  
where date_effective is null  
group by id_lecteur)
```

Une autre façon d'écrire cette requête plus loin
(en utilisant l'interaction entre agrégats et nulls)

```
select nom, prenom, count(*) as nb_livres  
from lecteur L left join emprunt E  
ON (L.id_lecteur = E.id_lecteur  
and date_effective is null)  
where id_e is null);
```

Vues et tables temporaires

Vues

- Les vues sont un mécanisme pour spécialiser une bases de données; utilisées également pour créer des tables temporaires virtuelles

Vues

- Parfois ce n'est pas souhaitable que tous les utilisateurs voient toutes les vraies tables stockées dans la BD
- **Exemple** : Considerer une personne qui a besoin de connaître les numéros de tous les prêts d'une banque, mais n'a pas besoin de voir les montants de ces prêts. Cette personne devrait uniquement voir une table décrite, en SQL, par :

```
(SELECT nom_client, num_prêt  
FROM Client C, Credit Cr  
WHERE C.id_client = Cr.id_client)
```

- Une **vue** fournit un mécanisme pour **cacher ou restructurer** les données pour certains utilisateurs.
- Une table qui n'est pas dans le schéma de la base de données, mais est rendue visible à un utilisateur en tant que "table virtuelle" est appelé une **vue**.

Définition de vue

- Une vue est définie en utilisant la commande **CREATE VIEW** de la forme :

CREATE VIEW *Nom_Vue* **AS** (< requête >)

où *Nom_Vue* est le nom de la vue et <requête> est toute requête SQL légale.

- Une liste de nom d'attributs (*att₁, ..., att_n*) après *Nom_Vue* est optionnelle
- Une fois la vue définie, *Nom_Vue* peut être utilisé dans les requêtes, à la place d'une table
- Définir une vue n'est pas la même chose que créer une nouvelle table du résultat de l'évaluation de <requête>: **le contenu d'une vue change automatiquement quand la base de données est modifiée**

Définition de vue : exemple

Créer une vue qui sélectionne les emprunts courants:

```
create view emprunts_courants as (  
    select date_emprunt, id_exemplaire,  
           id_lecteur, date_prevue  
    from emprunt  
    where date_effective is null  
)
```


Utilisation d'une vue

- Les lecteurs en retard de retour :

```
select distinct nom, prenom  
from emprunts_courants natural join lecteur  
where date_prevue < current_date;
```

- Une vue peut être utilisée dans les requêtes SELECT partout où on s'attend une table
- Une vue emprunts_courants n'est pas une table
(le résultat de la requête n'est pas stocké dans la base de données)
- Une vue est re-calculée à chaque fois qu'on a besoin de son contenu ⇒
emprunts_courants change automatiquement et donne toujours l'état actuel
(une vue) des emprunts en cours.

Les vues peuvent simplifier des requêtes complexes

Exemple

Trouver les départements avec salaire moyen maximal :

```
select département
from employe
group by department
having avg(salaire) >= ALL(
    select avg(salaire)
    from employe
    group by department
);
```

La requête en gris trouve les salaires moyens de tous les départements. Elle est implicitement utilisée deux fois dans la requête

La même requête en utilisant des vues

```
CREATE VIEW Salaires-Moyens (dpt, avg_sal) AS  
  (select departement, avg(salaire)  
   from employe group by department);
```

```
SELECT dpt  
FROM Salaires-Moyens  
WHERE avg_sal >= ALL  
      (SELECT avg_sal FROM Salaires-Moyens)
```

ou bien :

```
SELECT dpt  
FROM Salaires-Moyens  
WHERE avg_sal =  
      (SELECT MAX(avg_sal) FROM Salaires-Moyens)
```

Une syntaxe alternative: la clause **with**

```
WITH Salaires-Moyens (dpt, avg_sal) AS
  (select departement, avg(salaire)
   from employe group by departement)

SELECT dpt
FROM Salaires-Moyens
WHERE avg_sal =
      (SELECT MAX(avg_sal) FROM Salaires-Moyens)
```

Une syntaxe alternative: la clause **with**

```
WITH Salaires-Moyens (dpt, avg_sal) AS
(select departement, avg(salaire)
from employe group by departement)

SELECT dpt
FROM Salaires-Moyens
WHERE avg_sal =
      (SELECT MAX(avg_sal) FROM Salaires-Moyens)
```

Remarque: **Salaires-Moyens** définie par WITH est une **table temporaire**,
et **non pas une vue**

Difference entre table temporaire et vue :

- une vue, une fois créée est utilisable dans toutes les requêtes (sa définition augmente le schema de la base de données)
- une table temporaire est une façon de nommer une requête dans une autre requête, elle est utilisable uniquement dans la requête qui la définit (sa définition n'augmente pas le schema)

Une syntaxe alternative: la clause **with**

Exemple

Trouver les lecteurs qui ont le plus emprunté à la bibliothèque :

```
WITH NB_EMPRUNT(id_lecteur, nb) AS
    (select id_lecteur, count(*)
     from emprunt
     group by id_lecteur)
select nom, prenom
from lecteur natural join NB_EMPRUNT
where nb = (select max(nb) from NB_EMPRUNT);
```

Une syntaxe alternative: la clause **with**

Il est possible de définir plusieurs tables temporaires dans WITH :

```
WITH table1 as(select...),  
      table2 as (select ...)  
requête_select_principale;
```

Sous requêtes dans FROM

- Une autre façon de créer et utiliser une table temporaire : écrire une requête dans la clause FROM, à la place d'une table

Le nombre moyen d'emprunts par jour

```
select AVG (nb)
from ( select date_emprunt, count(*) as nb
      from emprunt
      group by date_emprunt
    ) as toto
```

Il est obligatoire de nommer une table définie par une requête dans FROM.

Sous requêtes dans FROM

- À utiliser quand la table temporaire sert une seule fois dans la requête. Sinon préférer WITH (ou CREATE VIEW)

Les dates où le nombre d'emprunts est supérieur au nombre moyen d'emprunts par jour

```
WITH Nombre_emprunts AS
( select date_emprunt, count(*) as nb
  from emprunt
  group by date_emprunt
)
select date_emprunt
from Nombre_emprunts
where nb > (select AVG(nb) from Nombre_emprunts)
```

Sous requêtes dans FROM

- À utiliser quand la table temporaire sert une seule fois dans la requête. Sinon préférer WITH (ou CREATE VIEW)

Les dates où le nombre d'emprunts est supérieur au nombre moyen d'emprunts par jour

Remarque : la même requête sans aucune forme de table temporaire est possible mais moins lisible

```
select date_emprunt
from emprunt
group by date_emprunt
having count (*) > (
    (select count(*) from emprunt) /
    (select count(distinct date) from emprunt)
)
```