Laboratoires de bases de données

Laboratoire n°3
Jointures

par Danièle BAYERS et Louis SWINNEN
Révision 2017 : Vincent Reip

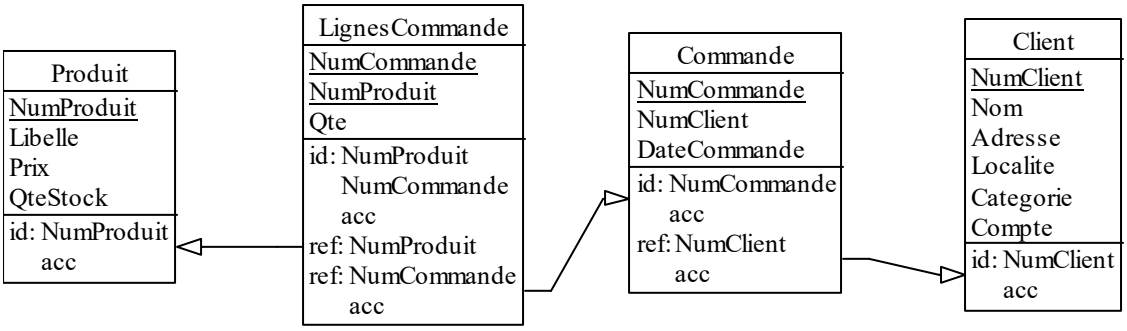
Octobre 2017

1. Objectif

L'objectif de ce laboratoire est d'aborder le concept de jointure dans son ensemble en commençant par le produit cartésien, ensuite nous abordons les jointures naturelles et conditionnelles.

2. La base de données

Dans les exemples, nous supposerons que nous disposons de la base de données suivante :



3. La jointure

La jointure permet l'utilisation de plusieurs tables dans une requête SQL. Mathématiquement, la jointure se définit comme un sous-ensemble du produit cartésien des relations (i.e. tables) listées.

La jointure peut être de type différent. On parle ainsi de **produit cartésien**, de **jointure naturelle** ou encore de **jointure conditionnelle**.

3.1 Le produit cartésien

Le *cross join* peut se représenter comme le produit cartésien complet entre les différentes tables. Tous les éléments sont donc repris dans le résultat.

Format :

```
SELECT attr1, attr2, …, attrn
FROM table1
CROSS JOIN table2
```

L'utilisation de CROSS JOIN est très limitée puisque le résultat obtenu n'est pas nécessairement significatif. Le SGBD créant des relations n'ayant pas nécessairement de sens.

Exemple :

```
SELECT * FROM client
CROSS JOIN commande
ORDER BY 1
```

Le résultat sera composé de l'ensemble des paires client – commande. Ainsi, dans ce résultat, chaque client sera associé à chaque commande. Le nombre de lignes dans le résultat sera donné par : **#client x #commande**

3.2 Les jointures naturelles

La jointure naturelle est une forme un peu particulière de jointure. En effet, en fonction des noms des attributs, le SGBD va automatiquement réaliser les liens entre les tables. Ainsi, lorsque les attributs présents dans les tables sélectionnées portent un nom identique, le SGBD considère qu'il s'agit de tables liées et construit le résultat en fonction de ces liens ainsi découverts.

On préfère toujours utiliser des jointures plus explicites. Les jointures naturelles peuvent, en cas d'évolution de la structure de la base de données, occasionner des dysfonctionnements dans des applications déjà existantes.

Format :

```
SELECT attr1, attr2, …, attrn
FROM table1
NATURAL JOIN table2
```

Exemple :

```
SELECT * FROM client
NATURAL JOIN commande
```

Sélectionne les clients ayant déjà passé au moins une commande. Le SGBD liera ici automatiquement les tables *client* et *commande* sur base du *numClient* présent dans chacune d'entre-elle.

3.3 Les jointures conditionnelles

Dans cette forme de jointure, nous allons exprimer une condition *d'égalité* permettant d'obtenir des enregistrements liés entre eux. En effet, en exploitant les clés étrangères liant les tables, nous obtenons des résultats exploitables.

Exemple : *obtenir le numéro et le libellé des produits ayant été commandés le 10/10/08*. Pour ce faire, il faut partir de la table produit, par les clés étrangères, obtenir les lignes de commandes reprenant ces produits et enfin les commandes liées à ces lignes de commande. Il reste à limiter le résultat à la date demandée.
Avec des requêtes ne portant que sur une seule table, il est impossible d'obtenir ce genre de résultats. Il est, par contre, très simple, d'écrire une seule requête donnant les produits demandés.

Format de la jointure (SQL-89) :

```
SELECT attribut1, attribut2, …, attributm
FROM table1, table2, …, tablen
WHERE condition1 AND condition2 AND … AND conditionn-1
```

Cette forme doit être abandonnée !

Format de la jointure (SQL-92) :

```
SELECT attribut1, attribut2, …, attributm
FROM table1
JOIN table2 ON condition1
…
JOIN tablen ON conditionn-1
```

La jointure permet d'écrire des requêtes impliquant plusieurs tables. Il est nécessaire lorsqu'on mentionne plusieurs tables dans une requête SQL d'inclure **les conditions de jointure**. Ces conditions expriment les liens existants entre les tables (très souvent au travers des clés

étrangères). Sur base de cette condition, le SGBD ne va garder que certains résultats jugés conformes par le programmeur.
Ainsi, la requête qui correspond à la demande exprimée plus haut peut s'écrire :

```
SELECT numProduit, libelle
FROM produit p
JOIN LignesCommande lc ON p.numProduit = lc.numProduit
JOIN Commande c ON c.numCommande = lc.numCommande
WHERE c.dateCommande = TO_DATE('10102008', 'DDMMYYYY') ;
```

À l'exécution de cette requête, on peut *imaginer* que le SGBD va créer tous les triplets possibles entre les tables *produit, LignesCommande* et *commande* (le produit cartésien de ces 3 tables). Une fois l'ensemble des résultats obtenus, le SGBD supprime les lignes qui ne satisfont pas aux conditions de jointure (égalité sur le numéro du produit entre la table *produit* et la table *lignesCommande*, égalité sur le numéro de *commande* entre la table *lignesCommande* et la table *commande*). Enfin, le SGBD supprime les enregistrements ne satisfaisant pas la condition `WHERE`.

Autres exemples :
Sélectionner les clients ayant commandés le produit numéro 123.
Dans cette condition, on exprime plusieurs liens entre des tables. Il faut sélectionner des clients ayant passés des commandes. Il faut, dans les commandes passées, garder uniquement celles dont au moins une ligne fait référence au produit 123.

```
SELECT c.numClient
FROM Commande c
JOIN LignesCommande lc ON  c.numCommande = lc.numCommande
WHERE lc.numProduit = 123
```

Si nous souhaitons afficher les informations de ce client particulier, il est nécessaire de faire une nouvelle jointure avec la table client. Ainsi, la requête s'exprimerait comme suit :

```
SELECT cli.nomClient, cli.Adresse, cli.Localite
FROM client cli
JOIN Commande c ON cli.numClient = c.numClient
JOIN LignesCommande lc ON c.numCommande = lc.numCommande
WHERE lc.numProduit = 123
```

Les jointures font parties des éléments très importants dans les bases de données. Grâce à elles, il est possible d'exploiter de grands dépôts de données en obtenant uniquement les résultats souhaités.

3.4 Les auto-jointures

Cette forme de jointure est parfois utilisée. Elle consiste à écrire une requête contenant une jointure désignant la même table (dans le JOIN) que celle de la requête elle-même (dans le FROM).

Cette forme particulière impose de donner des alias aux tables afin de pouvoir exprimer les conditions sans ambiguïté. La forme de cette jointure devient alors :

```
SELECT attr1, …, attrn
FROM table t1
JOIN table t2 ON t1.attrx = t2.attrx
WHERE t1.attry …
[ORDER BY …]
```

Dans cette requête, t1 désigne la table de la requête tandis que t2 désigne la table de la jointure.

Exemple :

```
SELECT c2.numCommande, c2.dateCommande
FROM commande c1
JOIN commande c2 ON c2.dateCommande > c1.dateCommande
WHERE c1.numCommande = 'C001'
```

Sélectionne les commandes dont la date de la commande est postérieure à celle de la commande C001.

3.5 Les jointures externes

Nous avons vu que les jointures permettaient, par la **condition de jointure**, de lier des tables entre-elles pour obtenir les enregistrements souhaités. Il est parfois important d'obtenir aussi les enregistrements pour lesquels il n'existe pas de correspondance dans l'autre table. Dans ces cas, les jointures externes sont utilisées.

Format :

```
SELECT attr1, …, attrn
FROM table1 t1
[FULL OUTER|LEFT OUTER|RIGHT OUTER] JOIN table2 t2 ON t1.attrx = t2.attry
WHERE t1.attry …
[ORDER BY …]
```

Dans cette forme, la jointure est réalisée entre la table *t1* et la table *t2*. En plus des résultats habituels suite à la jointure, nous obtenons les enregistrements n'ayant pas de correspondance. Dans le cas d'un `LEFT OUTER`, les enregistrements de la table *t1* n'ayant pas de correspondance (par la condition de jointure) dans la table *t2* sont inclus également dans le résultat. Dans le cas d'un `RIGHT OUTER`, les enregistrements de la table *t2* n'ayant pas de correspondance (par la condition de jointure) dans la table *t1* sont inclus dans le résultat.

Exemple (extrait de [1]) :

```
SELECT DISTINCT cli.numClient, cli.nom, com.dateCommande
FROM Client cli
LEFT OUTER JOIN Commande com ON cli.numClient = com.numClient
```

Dans cet exemple, on affiche les clients et leur date de commande. Si un client n'a pas commandé, il sera inclus dans le résultat mais sa date de commande vaudra NULL.

En plus de `LEFT OUTER` et `RIGHT OUTER`, SQL-2 définit `FULL OUTER` qui combine les deux options (`LEFT OUTER` et `RIGHT OUTER`) en même temps.

5. Stratégie pour l'écriture d'une jointure

Dans [1], l'auteur décrit la stratégie suivante pour l'écriture d'une jointure :

1. *« Déterminer les tables à mettre en jeu, les inclure dans la clause FROM (+JOIN).*
2. *Inclure les attributs à visualiser dans la clause SELECT*
3. *Déterminer s'il est nécessaire d'utiliser des jointures externes (et leur type)*
4. *Déterminer les conditions limitant la recherche. Les conditions portant sur les valeurs individuelles doivent figurer dans une clause WHERE*
5. *Préciser l'ordre d'apparition des tuples du résultat dans une clause ORDER BY. »*

Stratégie pour l'écriture d'une jointure (adapté de [1])