

Requêtes pivots sous MS-SQL Server

Débutant Avancé Expert



Complexité

par [Fabien Celaia](#)

Date de publication : 23.04.2006

Dernière mise à jour : 29.10.2006

Les requêtes pivots ont été largement popularisées par les QBE de Microsoft, que ce soit via MS-Query, MS-Excel ou MS-Access. Mais qu'en est-il de leur traitement via SQL en général, et MS-SQL en particulier ?

- I - Introduction
- II - Le point de vue SQL
- III - Le traitement via les outils Microsoft
 - III-A - MS-Excel
 - III-B - MS-Access
 - III-C - SQL Server
 - III-B-1 - Versions pré-2005 : diverses méthodes
 - III-C-2 - Versions pré-2000: transposition matricielle
 - III-C-3 - Version 2005 : PIVOT

I - Introduction

Les requêtes pivots ont été largement popularisées par les QBE de Microsoft, que ce soit via MS-Query, MS-Excel ou MS-Access. Elles ont de plus pris une importance considérable dans les outils d'aide à la décision, leur présentation agrégée permettant de condenser une information en une forme lisible, permettant l'analyse et la comparaison aisée de chiffres. Mais qu'en est-il de leur traitement via SQL en général, et MS-SQL en particuliers ?

Soit la table suivante

| Produit | Année | Vente |
|---------|-------|--------|
| A | 2005 | 12'000 |
| B | 2006 | 15'000 |
| C | 2005 | 1'000 |
| A | 2006 | 12'500 |
| C | 2004 | 850 |
| B | 2004 | 18'000 |
| A | 2004 | 10'000 |
| C | 2003 | 1'100 |
| A | 2003 | 9'000 |

Code pour la reproduction

```

Create table ventes
(produit char(1),
annee int,
vente float
)

insert into ventes values('A', 2005, 12000)
insert into ventes values('B', 2006, 15000)
insert into ventes values('C', 2005, 1000)
insert into ventes values('A', 2006, 12500)
insert into ventes values('C', 2004, 850)
insert into ventes values('B', 2004, 18000)
insert into ventes values('A', 2004, 10000)
insert into ventes values('C', 2003, 1100)
insert into ventes values('A', 2003, 9000)

```

La représentation recherchée est la suivante:

| | 2003 | 2004 | 2005 | 2006 |
|---|-------|--------|--------|--------|
| A | 9'000 | 10'000 | 12'000 | 12'500 |
| B | 0 | 18'000 | 0 | 15'000 |
| C | 1'100 | 850 | 1'000 | 0 |

II - Le point de vue SQL

Le SQL (Structured Query Language) est, comme son nom l'indique, un langage structuré permettant de traiter et de rendre l'information. Il sert donc de langue de communication entre l'application cliente et la base de données. Il a donc pour vocation de retourner les données stockées sur un SGBDR de manière brute, l'application cliente étant ensuite responsable de retourner l'information formatée comme attendue par l'utilisateur final.

Afin de simplifier la vie de leurs développeurs, la plupart des éditeurs de SGBDR ont très rapidement créés leur propre langage de programmation, permettant d'aller ainsi plus loin dans les traitements de l'information.

- Microsoft SQL Server= Transact-SQL
- Oracle = PL-SQL
- Sybase = Transact-SQL

Ces extensions du SQL permettent de faire des traitements sur les enregistrements, et même un peu de cosmétique (formatage, traitement de chaîne, conversion, ...). Ce faisant, on dénature un peu l'architecture multi-tiers: C'est en fait l'applicatif client qui devrait se charger de la cosmétique et de la mise en valeur de l'information.

III - Le traitement via les outils Microsoft

III-A - MS-Excel

Microsoft Excel, au travers de son assistant, a sans doute été l'un des pionniers de la popularisation des tables pivots de par son moteur d'analyse croisée dynamique. Rien de plus simple à utiliser.

[Cliquez ici pour afficher la démonstration sous Excel.](#)

III-B - MS-Access

Microsoft Access permet lui aussi de créer des tables pivots grâce à ses requêtes d'analyse croisée.

[Cliquez ici pour afficher la démonstration sous Access.](#)

... ce qui donne ce SQL... bien peu normé et quelque peu propriétaire:

```
TRANSFORM Sum(Vente) AS [Somme de vente]
SELECT Produit
FROM Ventes
GROUP BY Produit
PIVOT Annee;
```

III-C - SQL Server

III-B-1 - Versions pré-2005 : diverses méthodes

Il y a la méthode "bourrin" qui utilise une table temporaire et des mises à jours successives. Extrêmement coûteuse (chaque année supplémentaire génère un update supplémentaire), elle a le mérite d'être appréhendable par la majeure partie des débutants. C'est le parfait exemple d'un traitement séquentiel avec un langage typé ensembliste...

```
select distinct produit, 0 [2003], 0 [2004], 0 [2005], 0 [2006]
into #result
from Ventes

update #result
set [2003] = Vente
from Ventes v, #result r
where v.annee=2003
and v.produit=r.produit

update #result
set [2004] = Vente
from Ventes v, #result r
where v.annee=2004
and v.produit=r.produit

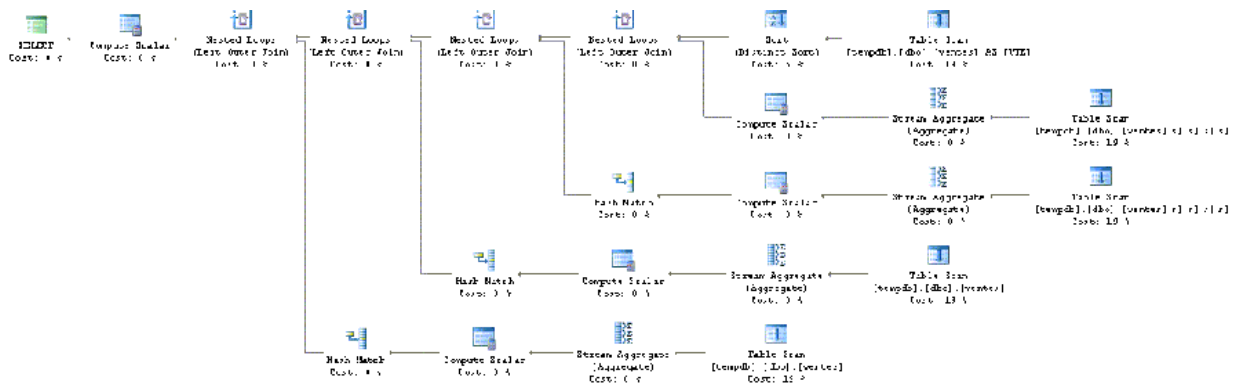
update #result
set [2005] = Vente
from Ventes v, #result r
where v.annee=2005
and v.produit=r.produit

update #result
set [2006] = Vente
from Ventes v, #result r
where v.annee=2006
and v.produit=r.produit
```

```
select * from #result
drop table #result
```

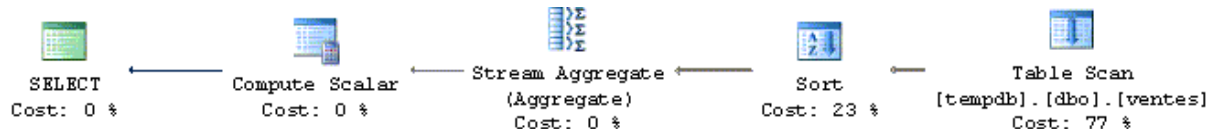
Ensuite vient la méthode que nous pourrions baptiser "propriétaire", qui utilise fortement les sous-requêtes.

```
SELECT produit,
  (SELECT SUM(vente)
   FROM Ventes
   WHERE annee = 2003
   AND produit = VTE.produit) AS [2003],
  (SELECT SUM(vente)
   FROM Ventes
   WHERE annee = 2004
   AND produit = VTE.produit) AS [2004],
  (SELECT SUM(vente)
   FROM Ventes
   WHERE annee = 2005
   AND produit = VTE.produit) AS [2005],
  (SELECT SUM(vente)
   FROM Ventes
   WHERE annee = 2006
   AND produit = VTE.produit) AS [2006]
FROM Ventes VTE
GROUP BY produit
```



Nettement plus élégante, ma préférence va à celle-ci...

```
select
  Product
, sum(case when annee=2003 then vente else 0 end) as [2003],
  sum(case when annee=2004 then vente else 0 end) as [2004],
  sum(case when annee=2005 then vente else 0 end) as [2005],
  sum(case when annee=2006 then vente else 0 end) as [2006],
  sum(vente ) as somme
from Ventes
group by produit
```



Il y a ensuite moyen de trafiquer une de ces méthodes pour la rendre quelque peu plus dynamique.

```
declare c cursor for select distinct annee from ventes order by annee
declare @sql varchar(2000), @col int
set @sql=''
```

```

open c
fetch c into @col
while @@FETCH_STATUS = 0
begin
select @sql=@sql+'SUM( case when annee='+ cast(@col as varchar(30)) +' then vente else 0
end) as [' + cast(@col as varchar(30)) +'],'
fetch c into @col
end
close c
deallocate c

exec ('SELECT Produit, '+ left(@sql,len(@sql)-1) +' from Ventes group by Produit' )

```

III-C-2 - Versions pré-2000: transposition matricielle

Plus compliquée, mais élégante et plus performante, la méthode de la transposition de matrice. Elle a l'avantage d'être portable sur quasi tous les SGBDR car elle n'utilise pas de SQL spécifique (tels que le CASE, les sous-requêtes, ...)

Elle utilise la spécificité qu'à le calcul matriciel de "retourner les données" lors de la multiplication par une matrice de 0 dont la diagonale est à 1.

J'en étais assez fier avant de trouver que cette méthode avait en fait déjà été exposée dans divers articles, dont l'excellent livre "[SQL Avancé de Joe Celko](#)", issue d'une idée de John M Baird... bien avant 2000 !

```

CREATE TABLE matrice(
    [Produit] char(1) PRIMARY KEY,
    [2003] int NOT NULL CONSTRAINT [DF_matrice2_2003] DEFAULT 0,
    [2004] int NOT NULL CONSTRAINT [DF_matrice2_2004] DEFAULT 0,
    [2005] int NOT NULL CONSTRAINT [DF_matrice2_2005] DEFAULT 0,
    [2006] int NOT NULL CONSTRAINT [DF_matrice2_2006] DEFAULT 0)

insert into matrice (Produit)
select distinct Produit
from Ventes

select matrice.Produit as Produit ,
sum ((1-abs (sign(2003-v.annee)))*vente) as [2003],
sum ((1-abs (sign(2004-v.annee)))*vente) as [2004],
sum ((1-abs (sign(2005-v.annee)))*vente) as [2005],
sum ((1-abs (sign(2006-v.annee)))*vente) as [2006]
from Ventes v, matrice where v.produit=matrice.produit
group by matrice.produit
go

```

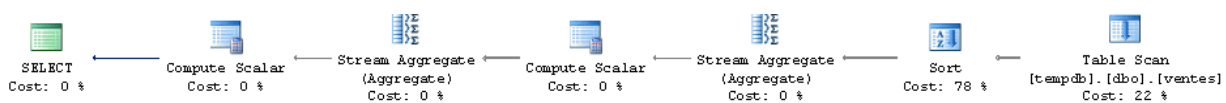
III-C-3 - Version 2005 : PIVOT

Bien que les méthodes vues ci-dessus marchent toujours, le T-SQL de Microsoft SQL Server 2005 accepte maintenant une notion de pivot:

```

SELECT Produit,
    SUM([2003]) AS [2003],
    SUM([2004]) AS [2004],
    SUM([2005]) AS [2005],
    SUM([2006]) AS [2006]
FROM Ventes
PIVOT (SUM(Vente) FOR annee IN ([2003], [2004], [2005], [2006])) AS Annees
GROUP BY Produit

```



... ce qui donne, en reprenant notre automatisation d'antan

```
declare c cursor for select distinct annee from ventes order by annee
declare @pivot varchar(200), @sum varchar(200), @sql nvarchar(1000), @col int

select @pivot='', @sum=''

open c
fetch c into @col
while @@FETCH_STATUS = 0
begin
select @sum = @sum + ' SUM(['+ cast(@col as varchar(30)) +']) AS ['+ cast(@col as
varchar(30)) +']',',
@pivot = @pivot + ' ['+ cast(@col as varchar(30)) +']', '
fetch c into @col
end
close c
deallocate c

set @sql = 'SELECT Produit, '+ left(@sum, len(@sum)-1)+
' FROM Ventes PIVOT (SUM(Vente)
FOR annee IN ('+ left(@pivot, len(@pivot)-1)+ ')) AS Annees
GROUP BY Produit'

exec(@sql)
```