Chapitre 5

Requêtes multi-tables

5.1 Introduction

Les requêtes utilisant plusieurs tables posent des problèmes spécifiques que nous allons aborder dans ce chapitre. Le premier problème concerne l'éventualité de plusieurs champs portant un même nom dans des tables différents. Le second consiste à devoir définir une « relation » entre ces différentes tables.

5.1.1 Noms qualifiés et alias

Lorsque plusieurs tables interviennent dans une requête, on peut utiliser une forme plus complète du nom d'un champ, un *nom qualifié*. Un tel nom se compose du nom de la table suivi d'un point, puis du nom du champ. L'utilisation de ce qualificatif est facultative, pour autant qu'il n'y ait aucune ambiguïté. Il reste que dans bien des cas, notamment pour les clés primaires et étrangères, on trouve des champs homonymes dans plusieurs tables. L'utilisation du qualificatif devient alors obligatoire. Il devient vite fastidieux de devoir recopier pour chaque champ le nom de la table, surtout dans les systèmes autorisant des requêtes sur des tables provenant de plusieurs bases. On a alors recours à des *alias*. En pratique, l'alias se réduit à l'initiale de la table ou, si nécessaire, aux premières lettres de son nom.

On entend par alias un nom alternatif donné à une table ou à une colonne. La norme SQL utilise les termes *correlative names* pour ce qui concerne les tables et *alias* pour les colonnes .

Pour ce qui concerne les alias de tables, ils viennent remplacer le vrai nom dans toute la requête pour les produits Microsoft et Oracle. InterBase continue à reconnaître le nom normal. Notons par avance que lorsqu'une table est employée plusieurs fois dans une requête multi-table, le recours à un alias devient obligatoire à partir de la deuxième occurrence de la table.

Alias pour table	SQL 2	Oracle	InterBase	SQL Server	mySQL
Précédé par	`AS` ou rien	rien	rien	`AS` ou rien	`AS ou rien`
Utilisable dans la requête en cours	oui	oblig.	facult.	oblig.	facult.

Exemples dans différents dialectes :

```
/* Oracle */
SELECT V.Client, V.Description
FROM Ventes V
```

```
/* InterBase */
SELECT V.Client, Ventes.Description
FROM Ventes V
/* mySQL */
SELECT V.Client, Ventes.Description
FROM Ventes AS V
```

5.1.2 Opérations relationnelles

L'algèbre relationnelle permet un certain nombre d'opérations formelles sur plusieurs relations. Voici la liste des opérations mettant en oeuvre deux ou plusieurs relations que nous allons évoquer dans les prochaines sections :

1. union
2. différence
3. intersection
4. produit cartésien
5. théta-produit
6. jointure naturelle
7. jointure extérieure
8. semi-jointure

5.2 Union

L'union consiste tout simplement à ajouter deux relations l'une à l'autre. Elle correspond à l'opération $\bigcup$ de la théorie des ensembles. Il va de soi qu'une union n'est concevable qu'entre deux relations qui ont la même structure (c'est-à-dire la même arité, le nombre d'éléments dans le tuple, et mêmes domaines pour chaque attribut). A priori, l'union peut s'appliquer à plus de deux tables, mais on procède dans ce cas-là à plusieurs unions.

IDVENTES	IDEM	CLIENT	DESCRIPTION	FOURNISSEUR	MONTANT
1	ADM	Thomas	Imprimante	IBM	9000
2	JB	Bernard	Souris	IBM	500
3	HB	Henri	Clavier	Apple	800
4	RS	Charles	Ecran	IBM	6000
5	LC	Henri	Imprimante	IBM	9000
6	JB	Thomas	Imprimante	IBM	9000
7	ADM	Charles	Carte Ethernet	Apple	1200

7 ligne(s) sélectionnée(s). **table VENTES**

IDVENTE	IDEMPLOYER	CLIENT	NATURE	FOURNISSEUR	MONTANT
3	HB	Henri	Clavier	Apple	800
5	LC	Henri	Imprimante	IBM	9000
213	XP	Benoît	Ecran	IBM	6000
216	VB	Fernand	Carthe Ethernet	Apple	1200
217	MS	Benoît	Imprimante	IBM	9000
218	NET	André	Imprimante	IBM	9000
225	MS	André	Souris	IBM	500

7 ligne(s) sélectionnée(s). **table VENTES2**

Il faut noter que le fait que *Ventes* et *Ventes2* contiennent chacune 7 tuples ne signifie nullement que leur union en comporte 14. En effet, les tuples communs ne peuvent figurer deux fois dans la relation, qui est, en dernière analyse, un ensemble. Pour des raisons de clarté, la relation ci-dessous a été triée. Mais l'ordre des tuples est en principe aléatoire.

IDVENTES	IDEM	CLIENT	DESCRIPTION	FOURNISSEUR	MONTANT
1	ADM	Thomas	Imprimante	IBM	9000
2	JB	Bernard	Souris	IBM	500
3	HB	Henri	Clavier	Apple	800
4	RS	Charles	Ecran	IBM	6000
5	LC	Henri	Imprimante	IBM	9000
6	JB	Thomas	Imprimante	IBM	9000
7	ADM	Charles	Carte Ethernet	Apple	1200
213	XP	Benoît	Ecran	IBM	6000
216	VB	Fernand	Carthe Ethernet	Apple	1200
217	MS	Benoît	Imprimante	IBM	9000
218	NET	André	Imprimante	IBM	9000
225	MS	André	Souris	IBM	500

12 ligne(s) sélectionnée(s).

UNION de Ventes et Ventes2

Voici la manière de créer cette opération en SQL :

```
SELECT * FROM Ventes
UNION
SELECT * FROM Ventes2 ;
```

L'union est une opération ensembliste incontournable si on veut réaliser des opérations sur deux tables. Elle est généralement implémentée car elle seule permet le mélange, dans les mêmes colonnes, de données provenant de deux tables. Il est important de noter, pour la suite, qu'une table virtuelle provenant d'une union n'est jamais modifiable : en effet nous ne disposons pas de l'information pour savoir dans quelle table effective nous devons reporter l'information.

Remarques d'utilisation

1. L'opérateur `UNION` s'applique en fait à des résultats de requêtes. On ne peut pas l'appliquer directement à des noms de tables.

   ```
   /* INCORRECT */
   SELECT  * FROM(Ventes UNION Ventes2) ;
   ```

2. Le nombre des colonnes et leur types doivent être équivalents. Dans la requête de l'exemple, tous les champs, sauf le quatrième, ont le même nom et le même type. Pour ce qui concerne le quatrième, la première table utilise le nom « Description » et la seconde le mot « Nature ». Seul le type importe. La requête suivante, bien qu'absurde, est légale parce que les champs *Client* et *Nature* sont de même type. Par contre, celles qui suivent enfreignent respectivement la règle sur le nombre et le type des colonnes.

   ```
   SELECT idEmploye,Client FROM Ventes
   UNION
   SELECT idEmploye,Nature FROM Ventes2;
   ```

IDEM	CLIENT
ADM	Charles
ADM	Thomas
HB	Clavier
HB	Henri
JB	Bernard
JB	Thomas
LC	Henri
LC	Imprimante
MS	Imprimante
MS	Souris
NET	Imprimante
RS	Charles
VB	Carthe Ethernet
XP	Ecran

```
SELECT idEmploye,Client,Description FROM Ventes
UNION
SELECT idEmploye,Nature FROM Ventes2
* ERREUR à la ligne 3 : ORA-01789:
le bloc interrogation contient un nombre incorrect de
colonnes résultat
```

```
SELECT idEmploye,Client FROM Ventes
UNION
SELECT idEmploye,Montant FROM Ventes2
* ERREUR à la ligne 3 : ORA-01790:
une expression doit être du même type que l'expression
qui␣lui␣correspond
```

Le fait qu'on s'intéresse uniquement au type des colonnes permet d'unir des tables dont les champs sont compatibles mais ne possèdent pas le même nom de colonne. En fait, c'est le nom ou l'alias du nom de la colonne de la première table qui donne son nom à la colonne du résultat.

3. Une union est en principe une opération ensembliste. Pour obtenir toutes les lignes des deux tables, on peut utiliser la commande `UNION ALL`, dont le résultat n'est plus une relation puisqu'il y a des doublons.

```
SELECT * FROM VENTES
UNION ALL
SELECT * FROM VENTES2
```

IDVENTES	IDEM	CLIENT	DESCRIPTION	FOURNISSEUR	MONTANT
1	ADM	Thomas	Imprimante	IBM	9000
2	JB	Bernard	Souris	IBM	500
3	HB	Henri	Clavier	Apple	800
4	RS	Charles	Ecran	IBM	6000
5	LC	Henri	Imprimante	IBM	9000
6	JB	Thomas	Imprimante	IBM	9000
7	ADM	Charles	Carte Ethernet	Apple	1200
3	HB	Henri	Clavier	Apple	800
5	LC	Henri	Imprimante	IBM	9000
213	XP	Benoît	Ecran	IBM	6000
216	VB	Fernand	Carthe Ethernet	Apple	1200
217	MS	Benoît	Imprimante	IBM	9000
218	NET	André	Imprimante	IBM	9000
225	MS	André	Souris	IBM	500

14 rows selected. **UNION ALL de Ventes et Ventes2**

4. Les requêtes insérées dans une union ne peuvent pas faire l'objet d'un tri (ce serait d'ailleurs inutile). Si l'expression `ORDER BY` figure dans une requête union, elle doit terminer la requête et s'applique au résultat de l'union.

```
Requête select 1
UNION [ALL|DISTINCT]
Requête select 2
```

5.3 Différence

La différence entre deux relations, qui sont « union-compatibles », contient tous les tuples qui sont dans la première relation et pas dans la seconde.

Comme en arithmétique, la différence est une opération non symétrique (7 - 4 n'est pas identique à 4 - 7). La différence entre *Ventes* et *Ventes2* comprend les éléments de *Ventes* qui ne sont pas dans *Ventes2*. Au contraire, la différence entre *Ventes2* et *Ventes* comprend les tuples de *Ventes2* qui ne sont pas dans *Ventes*. On remarquera que quel que soit l'ordre de l'opérateur, les éléments communs ne figurent jamais dans la relation résultante.

```
SELECT * FROM Ventes
EXCEPT
SELECT * FROM Ventes2 ;
```

IDVENTES	IDEM	CLIENT	DESCRIPTION	FOURNISSEUR	MONTANT
1	ADM	Thomas	Imprimante	IBM	9000
2	JB	Bernard	Souris	IBM	500
4	RS	Charles	Ecran	IBM	6000
6	JB	Thomas	Imprimante	IBM	9000
7	ADM	Charles	Carte Ethernet	Apple	1200

```
SELECT * FROM Ventes2
EXCEPT
SELECT * FROM Ventes ;
```

IDVENTE	IDEMPLOYER	CLIENT	NATURE	FOURNISSEUR	MONTANT
213	XP	Benoît	Ecran	IBM	6000
216	VB	Fernand	Carthe Ethernet	Apple	1200
217	MS	Benoît	Imprimante	IBM	9000
218	NET	André	Imprimante	IBM	9000
225	MS	André	Souris	IBM	500

La plupart des SGBDR courants n'autorisent pas les opérations de différence. Oracle est un des rares à le proposer, mais sous le nom non standard de `MINUS`. Avec un peu de réflexion, on arrive cependant à proposer une requête qui fournit le même résultat[1].

```
Requête select 1
EXCEPT
Requête select 2
```

5.4 Intersection

L'intersection de deux relations, qui sont encore « union-compatibles », comprend tous les tuples qui sont à la fois dans les deux relations, cela correspond à l'opérateur ensembliste $\cap$. Contrairement à la différence, l'ordre des relations intervenant dans l'intersection est sans importance.

IDVENTES	IDEM	CLIENT	DESCRIPTION	FOURNISSEUR	MONTANT
3	HB	Henri	Clavier	Apple	800
5	LC	Henri	Imprimante	IBM	9000

La traduction SQL de cette opération est la suivante :

```
SELECT * FROM Ventes
INTERSECT
SELECT * FROM Ventes2 ;
```

Comme la différence, l'intersection est rarement implémentée dans les SGBDR[2]. Oracle l'offre également.

```
Requête select 1
INTERSECT
Requête select 2
```

[1] Il serait dommage de priver les étudiants du plaisir d'en chercher eux-mêmes l'écriture. La recherche d'une équivalence de la différence sera donc proposée comme exercice.

[2] Un nouvel exercice en perspective...

5.5 Produit cartésien

IDVENTES	IDEM	CLIENT	DESCRIPTION	FOURNISSEUR	MONTANT
1	ADM	Thomas	Imprimante	IBM	9000
2	JB	Bernard	Souris	IBM	500
3	HB	Henri	Clavier	Apple	800
4	RS	Charles	Ecran	IBM	6000
5	LC	Henri	Imprimante	IBM	9000
6	JB	Thomas	Imprimante	IBM	9000
7	ADM	Charles	Carte Ethernet	Apple	1200

7 ligne(s) sélectionnée(s).

table VENTES

IDEMPLOYE	PRENOM	NOM	DNAISS	DENG	NVOITURE
HB	Hubert	Billen	09/07/58	12/05/99	
LC	Laurence	Comhaire	15/03/78	01/09/05	4
ADM	Anne	De Mey	12/10/69	19/05/05	
JB	Jacques	Bawin	01/02/60	01/09/94	1
RS	Roland	Schyns	24/12/68	05/12/01	3

table Employes

Le produit cartésien de deux relations consiste à créer de nouveaux tuples, dont le nombre de colonnes est égal à la somme du nombre des colonnes des relations de base. On combine chaque tuple de la première relation avec chacun des tuples de la deuxième relation. Il s'agit d'une opération qui produit normalement beaucoup de lignes. Dans notre cas, en combinant la relation Ventes (7 tuples) avec la relation Employes (5 tuples), on obtient un produit cartésien de 35 lignes.

IDVENTES	IDEM	CLIENT	DESCRIPTION	FOURNISSEUR	MONTANT	IDEMP	PRENOM	NOM	DNAISS	DENG	NVOITURE
1	ADM	Thomas	Imprimante	IBM	9000	HB	Hubert	Billen	09-JUL-58	12-MAY-99	
2	JB	Bernard	Souris	IBM	500	HB	Hubert	Billen	09-JUL-58	12-MAY-99	
3	HB	Henri	Clavier	Apple	800	HB	Hubert	Billen	09-JUL-58	12-MAY-99	
4	RS	Charles	Ecran	IBM	6000	HB	Hubert	Billen	09-JUL-58	12-MAY-99	
5	LC	Henri	Imprimante	IBM	9000	HB	Hubert	Billen	09-JUL-58	12-MAY-99	
6	JB	Thomas	Imprimante	IBM	9000	HB	Hubert	Billen	09-JUL-58	12-MAY-99	
7	ADM	Charles	Carte Ethernet	Apple	1200	HB	Hubert	Billen	09-JUL-58	12-MAY-99	
1	ADM	Thomas	Imprimante	IBM	9000	LC	Laurence	Comhaire	15-MAR-78	01-SEP-05	4
2	JB	Bernard	Souris	IBM	500	LC	Laurence	Comhaire	15-MAR-78	01-SEP-05	4
3	HB	Henri	Clavier	Apple	800	LC	Laurence	Comhaire	15-MAR-78	01-SEP-05	4
4	RS	Charles	Ecran	IBM	6000	LC	Laurence	Comhaire	15-MAR-78	01-SEP-05	4
5	LC	Henri	Imprimante	IBM	9000	LC	Laurence	Comhaire	15-MAR-78	01-SEP-05	4
6	JB	Thomas	Imprimante	IBM	9000	LC	Laurence	Comhaire	15-MAR-78	01-SEP-05	4
7	ADM	Charles	Carte Ethernet	Apple	1200	LC	Laurence	Comhaire	15-MAR-78	01-SEP-05	4
1	ADM	Thomas	Imprimante	IBM	9000	ADM	Anne	De Mey	12-OCT-69	19-MAY-05	
2	JB	Bernard	Souris	IBM	500	ADM	Anne	De Mey	12-OCT-69	19-MAY-05	
3	HB	Henri	Clavier	Apple	800	ADM	Anne	De Mey	12-OCT-69	19-MAY-05	
4	RS	Charles	Ecran	IBM	6000	ADM	Anne	De Mey	12-OCT-69	19-MAY-05	
5	LC	Henri	Imprimante	IBM	9000	ADM	Anne	De Mey	12-OCT-69	19-MAY-05	
6	JB	Thomas	Imprimante	IBM	9000	ADM	Anne	De Mey	12-OCT-69	19-MAY-05	
7	ADM	Charles	Carte Ethernet	Apple	1200	ADM	Anne	De Mey	12-OCT-69	19-MAY-05	
1	ADM	Thomas	Imprimante	IBM	9000	JB	Jacques	Bawin	01-FEB-60	01-SEP-94	1
2	JB	Bernard	Souris	IBM	500	JB	Jacques	Bawin	01-FEB-60	01-SEP-94	1
3	HB	Henri	Clavier	Apple	800	JB	Jacques	Bawin	01-FEB-60	01-SEP-94	1

Page suivante

Le produit cartésien, outre le fait qu'il réussit généralement à saturer la machine sur laquelle on l'exécute, produit des tuples sans significations. Dans la relation ci-dessus, seules

certaines lignes ont une signification : ce sont celles où l'identifiant de l'employé est identique dans les deux moitiés de la ligne.

Il reste une petite difficulté à résoudre. Le produit cartésien que nous venons d'appliquer produit une table dont deux colonnes portent le même nom. Dans la pratique, c'est inacceptable. Le SGBDR devra trouver un moyen d'éviter cela, généralement en donnant un autre nom à la colonne répétitive.

Le produit cartésien est souvent obtenu par erreur. Sa traduction SQL est simple :

```
SELECT *
FROM Ventes, Employes ;
```

Bien qu'à proprement parler, les produits cartésiens ne soient pas des jointures. la norme SQL2 propose la forme `CROSS JOIN`, ignorée par InterBase mais non par Oracle.

```
SELECT *
FROM Employes CROSS JOIN Ventes;
```

```
SELECT Liste de champs
FROM Table1 CROSS JOIN
Table2
```

Variante si `CROSS JOIN` n'est pas disponible :

```
SELECT Liste de champs
FROM Table1 , Table2
```

5.6 Théta-produits et jointures

On appelle «théta-produit», un produit cartésien suivi d'une sélection. Comme tels, les théta-produits ne sont pas toujours intéressants. On les retrouve sous différentes formes nommées jointures.

```
SELECT Liste de champs
FROM Table1 , Table2
WHERE Condition
```

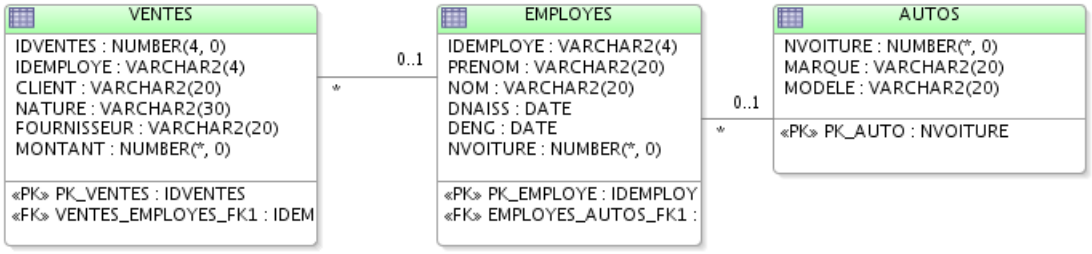
Une jointure s'emploie entre deux ou plusieurs tables (ou éventuellement une même table utilisée plusieurs fois). En général, sauf dans le cas du produit cartésien, on assortit la jointure d'une spécification sur la manière d'opérer la jonction. Les deux premières versions de la norme SQL diffèrent sensiblement dans la manière d'exprimer les jointures.

La norme 1 ne reconnaît pas explicitement les jointures. Une jointure est simplement une requête comportant plusieurs tables après l'expression `FROM`. Pour préciser la manière d'opérer la jointure, on utilise une condition introduite par `WHERE`. Notons que cette notation peut continuer à s'employer sur les SGDB qui suivent la norme 2. L'utilisation de plusieurs noms de tables et d'une expression `WHERE` ne permet cependant pas de réaliser toutes les formes de jointures. Les différents systèmes proposent alors des formalismes particuliers et incompatibles entre eux.

La norme 2 fournit l'expression `JOIN` utilisée conjointement avec d'autres mots pour spécifier les différents types de jointures. La norme SQL2 concernant les jointures n'est pas toujours implémentée complètement sur tous les systèmes. Il faudra donc se référer à la documentation du SGBD pour savoir quels formulations syntaxiques sont autorisées. Notons que dans la plupart des cas, les syntaxes non reconnues peuvent facilement se voir substituer d'autres formulations.

Présentation des tables exemples

Nous allons utiliser trois tables pour nos exemples : ces tables fonctionnent comme une petite base de données dans laquelle nous établissons des liens entre des ventes, des employés et des voitures (non concernées par les ventes).



Pour chaque vente, on donne l'identifiant de l'employé qui a réalisé la vente. En outre, certains employés ont une voiture, répertoriée dans la table Auto.

IDVENTES	IDEM	CLIENT	DESCRIPTION	FOURNISSEUR	MONTANT
1	ADM	Thomas	Imprimante	IBM	9000
2	JB	Bernard	Souris	IBM	500
3	HB	Henri	Clavier	Apple	800
4	RS	Charles	Ecran	IBM	6000
5	LC	Henri	Imprimante	IBM	9000
6	JB	Thomas	Imprimante	IBM	9000
7	ADM	Charles	Carte Ethernet	Apple	1200

7 ligne(s) sélectionnée(s).

table VENTES

IDEMPLOYE	PRENOM	NOM	DNAISS	DENG	NVOITURE
HB	Hubert	Billen	09/07/58	12/05/99	
LC	Laurence	Comhaire	15/03/78	01/09/05	4
ADM	Anne	De Mey	12/10/69	19/05/05	
JB	Jacques	Bawin	01/02/60	01/09/94	1
RS	Roland	Schyns	24/12/68	05/12/01	3

table Employes

NVOITURE	MARQUE	MODELE
1	Opel	Corsa
2	Renault	Mégane
3	Opel	Astra
4	Ford	Fiesta
5	Renault	Safrane
6	Ford	Escort
7	Renault	Mégane
8	Jaguar	Type E

8 ligne(s) sélectionnée(s).

table Autos

5.7 Jointure interne

Une jointure interne est un théta-produit dans lequel la sélection impose une égalité sur des attributs identiques. Pour retrouver les lignes significatives du produit cartésien vu plus haut, on utilisera une jointure interne sur le champ idEmploye.

IDEMPLOYE	IDVENTES	CLIENT	NATURE	FOURNISSEUR	MONTANT	PRENOM	NOM	DNAISS	DENG	NVOITURE
ADM	1	Thomas	Imprimante	IBM	9000	Anne	De Mey	12/10/69	19/05/05	
JB	2	Bernard	Souris	IBM	500	Jacques	Bawin	01/02/60	01/09/94	1
HB	3	Henri	Clavier	Apple	800	Hubert	Billen	09/07/58	12/05/99	
RS	4	Charles	Ecran	IBM	6000	Roland	Schyns	24/12/68	05/12/01	3
LC	5	Henri	Imprimante	IBM	9000	Laurence	Comhaire	15/03/78	01/09/05	4
JB	6	Thomas	Imprimante	IBM	9000	Jacques	Bawin	01/02/60	01/09/94	1
ADM	7	Charles	Carte Ethernet	Apple	1200	Anne	De Mey	12/10/69	19/05/05	

7 ligne(s) sélectionnée(s).

SQL permet d'exprimer une jointure interne comme un produit cartésien complété par une sélection :

```
SELECT Employes.*, Ventes.*
FROM Employes, Ventes
WHERE Employes.idEmploye = Ventes.idEmploye;
```

Une nouvelle norme, apparue plus tardivement, permet de formuler la jointure de manière explicite, au moyen de l'opérateur `INNER JOIN...ON`.

```
SELECT Employes.*, Ventes.*
FROM Employes INNER JOIN Ventes
ON Employes.idEmploye = Ventes.idEmploye;
```

5.7.1 Jointures internes simples

Une jointure interne est une jointure dans laquelle les valeurs des colonnes jointes sont comparées à l'aide d'un opérateur de comparaison. Elle correspond à une opération fondamentale des SGBDR. En effet, la répartition des données dans plusieurs tables nécessite l'utilisation des jointures pour recomposer les données telles qu'elles apparaissent dans la réalité. Dans le cas de deux tables, on aura une relation parent-enfant qui fera correspondre à chaque ligne d'une table une ou plusieurs lignes d'une autre table. Dans notre magasin d'ordinateurs, la table des ventes comporte un champ `idEmploye`, qui correspond au numéro du vendeur ayant réalisé la vente.

Afficher la liste des ventes avec le nom du client, la description du produit et le nom du vendeur

```
SELECT Client, Description AS Article, Nom AS Vendeur
FROM Ventes V
INNER JOIN Employes E ON V.idEmploye=E.idemploye
```

CLIENT	ARTICLE	VENDEUR
Thomas	Imprimante	De Mey
Bernard	Souris	Bawin
Henri	Clavier	Billen
Charles	Ecran	Schyns
Henri	Imprimante	Comhaire
Thomas	Imprimante	Bawin
Charles	Carte Ethernet	De Mey

La liaison entre les deux tables se fait par l'intermédiaire du champ `idEmploye` (qu'on retrouve dans chacune d'elles et qui contient le numéro du vendeur). Comme les champs correspondants sont homonymes, une pratique courante, on est obligé ici d'utiliser un nom qualifié, en lui préfixant le nom de la table d'où il provient ou son alias.

Le champ idEmploye apparaît dans les deux tables et contient en fait le « numéro du vendeur ». Dans la table *Employes*, le champ idEmploye fait office de *clé primaire*, cela implique que chaque ligne contient une valeur de idEmploye différente des autres. Par contre dans la table *Ventes*, la clé est dite *étrangère*. Ses valeurs ne sont pas uniques (un vendeur peut réaliser plusieurs ventes). Elle nous permet de faire correspondre un vendeur unique à chaque vente. Cette unicité est garantie par le fait que le champ idEmploye est clé primaire dans l'autre table.

IDEMPLOYE	PRENOM	NOM	DNAISS	DENG	NVOITURE
HB	Hubert	Billen	09/07/58	12/05/99	
LC	Laurence	Comhaire	15/03/78	01/09/05	4
ADM	Anne	De Mey	12/10/69	19/05/05	
JB	Jacques	Bawin	01/02/60	01/09/94	1
RS	Roland	Schyns	24/12/68	05/12/01	3

table Employes

L'utilisation d'une jointure n'impose pas que les champs portent le même nom. Dans la conception d'une base de données, il est souvent conseillé de mettre le même nom, si c'est possible.

Malheureusement, la jointure ne correspond pas toujours à une opération significative :

- si les champs utilisés pour la jointure n'est pas clé primaire dans une table, on va générer des lignes plus nombreuses, mais impossibles à interpréter ;
- il faut que les données de la réalité correspondent : on peut parfaitement faire une jointure entre le numéro de la vente de la table *Ventes* et le numéro de la voiture de la table *Autos*. Le résultat obtenu n'aura aucun sens.
- si une donnée de la clé étrangère ne correspond à aucune donnée de la clé primaire, la ligne qui la contient n'apparaîtra pas dans la jointure. Cela correspondrait par exemple à une vente réalisée par un vendeur inconnu. On dira que la ligne est *orpheline*. C'est une erreur grave dans les données : il devient impossible de recomposer l'information complète. On parle de *rupture d'intégrité référentielle*. Nous verrons plus loin qu'il est possible d'empêcher l'apparition des lignes orphelines en définissant des contraintes. En réalité, une clé étrangère est un champ pour lequel on a défini cette contrainte d'intégrité.

Il existe des variantes, notamment implémentées dans Oracle, faisant disparaître le mot `INNER`, ou utilisant le mot `USING` pour marquer les champs homonymes utilisés[3] :

```
SELECT idVente,  Client, Description, Nom
FROM Ventes V JOIN Employes E ON V.idEmploye = E.idEmploye

SELECT NV,  Client, Description, Nom
FROM Ventes V INNER JOIN Employes USING (idEmploye)
```

```
SELECT Liste de champs
FROM Table1 INNER JOIN Table2
ON Champ Table 1 = Champ Table 2
```

```
SELECT Liste de champs
FROM Table1 INNER JOIN Table2
USING ( Liste de champs communs )
```

Pour obtenir le même résultat en SQL1, on doit recourir à un thêta-produit, qui est, rappelons-le, un produit cartésien assorti d'une sélection. Cette requête fonctionnera sur tous les systèmes. Elle présente évidemment l'inconvénient de mélanger la condition de jointure exprimée dans la clause `WHERE` avec une éventuelle condition de restriction (il faudra utiliser le connecteur `AND`).

```
SELECT idVente,  Client, Description, Nom
FROM Ventes V , Employes E
WHERE V.idEmploye = E.idEmploye
```

```
SELECT Liste de champs
FROM Table1 , Table2
WHERE Champ Table 1 = Champ Table 2
```

(1) equi-joins : Dans la plupart des cas, les jointures internes sont basées sur des égalités. La littérature anglo-saxonne les appelle des *equi-joins*. Elles correspondent aux opérations courantes.

[3]Cette variante n'est pas reconnue par InterBase, ni par SQL-Server. Notons qu'il peut y avoir plusieurs champs impliqués. On les sépare alors par des virgules.

(2) non-equi joins : Le comparateur n'est pas toujours l'égalité. On ne trouvera pas d'exemple simple d'utilisation des opérateurs <>, < ou <=. Par contre, `BETWEEN` permet de résoudre facilement des questions apparemment sans solution. Pour afficher les grades obtenus par des étudiants en fonction des résultats, on pourra disposer d'une table *Grades* :

GRADE	MIN	MAX
Satisfaction	6	6
Distinction	7	7
Grande distinction	8	8
La plus grande distinction	9	10
Echec	0	5

Afficher les grades à l'examen

```
SELECT Nom, Examen,Grade  FROM Etudiants
INNER JOIN Grades
ON Examen BETWEEN Min AND MAX
```

NOM	EXAMEN	GRADE
GALLINA	4	Echec
DESIRONT	4	Echec
LEBE DESSARD	4	Echec
MATARAZZI	4	Echec
BOULAICH	4	Echec
FAUCHE	4	Echec
FRANCOIS	8	Grande distinction
ROORDA	8	Grande distinction
ABDULLAH	8	Grande distinction
CAPIAU	8	Grande distinction
PIERANTONIO	8	Grande distinction

33 rows selected.

5.7.2 jointures naturelles

La norme SQL 2 introduit une dernière manière de réaliser la jointure de deux tables : une jointure naturelle. Cette jointure utilise les champs communs aux deux tables pour réaliser la jointure (d'où son nom de « naturelle »).

```
SELECT Liste de champs
FROM Table1 NATURAL JOIN Table2
```

Afficher la liste des ventes avec le nom du client, la description du produit et le nom du vendeur

```
SELECT idEmploye,Client, Description, Nom
FROM Ventes V NATURAL JOIN Employes E
```

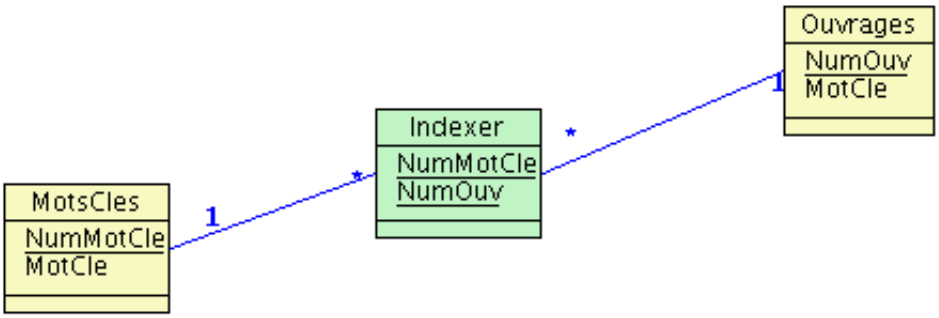
IDEMP	CLIENT	DESCRIPTION	NOM
ADM	Thomas	Imprimante	De Mey
JB	Bernard	Souris	Bawin
HB	Henri	Clavier	Billen
RS	Charles	Ecran	Schyns
LC	Henri	Imprimante	Comhaire
JB	Thomas	Imprimante	Bawin
ADM	Charles	Carte Ethernet	De Mey

Très séduisante au premier abord, cette méthode peut provoquer des résultats désastreux. Si un programmeur utilise cette syntaxe et qu'on ajoute par la suite des champs homonymes sans rapport avec la jointure, ces nouveaux champs viendront perturber l'opération. Un exemple simple serait un champ de contrôle placé dans chaque table pour mémoriser la date de la dernière modification. On peut certifier que plus aucune jointure naturelle ne fonctionnerait. C'est donc une variante commode à réserver au travail sur une console, **mais à ne jamais utiliser en programmation**, que ce soit dans une application externe ou dans un *trigger* ou une procédure exécutée sur le serveur.

5.7.3 jointures multiples

Il est parfaitement possible d'utiliser plus de deux tables pour réaliser une requête.

Voici un exemple classique permettant de faire la liaison entre une liste de mots-clés et une liste d'ouvrages indexés à l'aide de ces mots clés. On a typiquement deux tables d'entités (*MotsCles* et *Ouvrages*) et une table de relation (*Indexer*). La clé primaire *NumMotCle* de *Motscles* est clé étrangère dans *Indexer* et de même que la clé primaire *NumOuv* d'*Ouvrages*.



Donner la liste des ouvrages concernant les accidents de travail

```
SELECT Titre
FROM MotsCles M
   INNER JOIN Indexer I ON M.NumMotCle = I.NumMotCle
   INNER JOIN Ouvrages O ON L.NumOuv = O.NumOuv
WHERE MotCle="ACCIDENT␣DU␣TRAVAIL"
ORDER BY 1;
```

5.7.4 autojointures

Les autojointures ne sont rien d'autre que des jointures dans lesquelles une même table figure deux fois. On est obligé dans ce cas d'utiliser un alias pour distinguer les champs qui

appartiennent à la première occurrence de la table de ceux de la deuxième. Comme exemple, nous reprendrons la table *Emp* de Scott du chapitre précédent. Le champ *Mgr* contient le numéro du chef de service de chaque employé. Comme les deux tables portent le même nom, l'usage d'au moins un alias devient obligatoire.

Donner la liste des employés, avec le nom de leur chef de service.

```
SELECT E.EmpNo, E. EName, E. MGR, Chef.EName NomMgr
FROM Emp E INNER JOIN Emp Chef
ON E.MGR=Chef.EmpNo
```

EMPNO	ENAME	MGR	NOMMGR
7369	SMITH	7902	FORD
7499	ALLEN	7698	BLAKE
7521	WARD	7698	BLAKE
7566	JONES	7839	KING
7654	MARTIN	7698	BLAKE
7698	BLAKE	7839	KING
7782	CLARK	7839	KING
7788	SCOTT	7566	JONES
7844	TURNER	7698	BLAKE
7876	ADAMS	7788	SCOTT
7900	JAMES	7698	BLAKE
7902	FORD	7566	JONES
7934	MILLER	7782	CLARK

Remarquons que Mr King (employé 7839), qui n'a pas de chef, ne figure pas dans le résultat.

5.8 Jointure externe

Pour illustrer la notion de jointure extérieure, nous allons utiliser la table *Autos*.

IDEMPLOYE	PRENOM	NOM	DNAISS	DENG	NVOITURE
HB	Hubert	Billen	09/07/58	12/05/99	
LC	Laurence	Comhaire	15/03/78	01/09/05	4
ADM	Anne	De Mey	12/10/69	19/05/05	
JB	Jacques	Bawin	01/02/60	01/09/94	1
RS	Roland	Schyns	24/12/68	05/12/01	3

table Employes

NVOITURE	MARQUE	MODELE
1	Opel	Corsa
2	Renault	Mégane
3	Opel	Astra
4	Ford	Fiesta
5	Renault	Safrane
6	Ford	Escort
7	Renault	Mégane
8	Jaguar	Type E

8 ligne(s) sélectionnée(s).

table Autos

La norme SQL2

Une jointure interne des *Employes* et *Autos* nous donne une table contenant l'association de chaque conducteur avec sa voiture. De manière évidente, seules y figurent les voitures qui ont un propriétaire et seuls les employés qui ont une voiture.

```
SELECT Prenom, Nom, Marque, Modele
FROM Employes INNER JOIN Autos USING (NVoiture);
```

PRENOM	NOM	MARQUE	MODELE
Laurence	Comhaire	Ford	Fiesta
Jacques	Bawin	Opel	Corsa
Roland	Schyns	Opel	Astra

Il peut s'avérer intéressant de voir la liste des employés avec leur voiture, y compris ceux qui n'en ont pas et, à l'inverse, la liste des voitures avec leur propriétaires, y compris celles qui n'en n'ont pas. On y parvient au moyen de jointures extérieures.

Une jointure extérieure est nécessairement asymétrique :

– soit on privilégie la première relation (celle qui est à gauche) et toutes les lignes qu'elle contient apparaissent dans la relation résultante. On obtient dans notre exemple, la liste des employés avec précision du véhicule qu'ils utilisent . On constate alors que Hubert et Anne n'ont pas de voiture.

IDEMPLOYE	PRENOM	NOM	DNAISS	DENG	NVOITURE	NVOITURE	MARQUE	MODELE
HB	Hubert	Billen	09/07/58	12/05/99				
LC	Laurence	Comhaire	15/03/78	01/09/05	4	4	Ford	Fiesta
ADM	Anne	De Mey	12/10/69	19/05/05				
JB	Jacques	Bawin	01/02/60	01/09/94	1	1	Opel	Corsa
RS	Roland	Schyns	24/12/68	05/12/01	3	3	Opel	Astra

Traduction SQL :

```
SELECT Employes.*, Autos.*
FROM Employes LEFT JOIN Autos
ON Employes.NVoiture = Autos.NVoiture ;
```

– soit on privilégie la seconde (celle qui est à droite) et toutes les lignes de la deuxième relation figurent dans la relation finale. Notre exemple donne la liste des voitures avec leur propriétaire éventuel. Seuls les véhicules 1, 3 et 4 ont un propriétaire connu.

IDEMPLOYE	PRENOM	NOM	DNAISS	DENG	NVOITURE	NVOITURE	MARQUE	MODELE
JB	Jacques	Bawin	01/02/60	01/09/94	1	1	Opel	Corsa
						2	Renault	Mégane
RS	Roland	Schyns	24/12/68	05/12/01	3	3	Opel	Astra
LC	Laurence	Comhaire	15/03/78	01/09/05	4	4	Ford	Fiesta
						5	Renault	Safrane
						6	Ford	Escort
						7	Renault	Mégane
						8	Jaguar	Type E

Traduction SQL :

```
SELECT Employes.*, Autos.*
FROM Employes RIGHT JOIN Autos
ON Employes.NVoiture =  Autos.NVoiture;
```

Dans les deux cas, un grand nombre de champs nuls figurent dans la relation obtenue. Nous en tirerons parti plus loin.

La norme SQL2 propose donc de représenter ces jointures à l'aide des expressions `LEFT OUTER JOIN` et `RIGHT OUTER JOIN`. Le mot `OUTER` est facultatif et pas souvent employé.

```
SELECT Liste de champs
FROM Table1 LEFT [OUTER] JOIN Table2
ON Champ Table 1 = Champ Table 2
```

```
SELECT Liste de champs
FROM Table1 RIGHT [OUTER] JOIN Table2
ON Champ Table 1 = Champ Table 2
```

L'appellation gauche ou droite dépend bien sûr de l'ordre dans lequel on énumère les relations constitutives de la jointure. Les deux requêtes suivantes donnent donc des résultats équivalents :

```
SELECT Champ1, Champ2
FROM Table1 LEFT JOIN Table2 USING (Champ3);
SELECT Champ1, Champ2
FROM Table2 RIGHT JOIN Table1 USING (Champ3);
```

Il existe également une jointure totale, qui reprend tous les éléments des deux tables :

IDEMPLOYE	PRENOM	NOM	DNAISS	DENG	NVOITURE	NVOITURE	MARQUE	MODELE
HB	Hubert	Billen	09/07/58	12/05/99				
LC	Laurence	Comhaire	15/03/78	01/09/05	4	4	Ford	Fiesta
ADM	Anne	De Mey	12/10/69	19/05/05				
JB	Jacques	Bawin	01/02/60	01/09/94	1	1	Opel	Corsa
RS	Roland	Schyns	24/12/68	05/12/01	3	3	Opel	Astra
						5	Renault	Safrane
						8	Jaguar	Type E
						2	Renault	Mégane
						6	Ford	Escort
						7	Renault	Mégane

```
SELECT Liste de champs
FROM Table1 FULL JOIN Table2
ON Champ Table 1 = Champ Table 2
```

Simulation des jointures externes dans SQL1

La norme SQL1 ne propose pas de moyen officiel pour réaliser les jointures externes. Il faut donc réaliser les opérations par des moyens détournés[4]. Cependant, la plupart des SGBD ont proposé des moyens plus directs, mais malheureusement particuliers à chacun. Je propose de montrer ici comment les versions antérieures d'Oracle se proposaient de réaliser les trois types de jointures. Elles fonctionnent toujours sur nos systèmes 10 et 11.

```
/*Jointure gauche */
SELECT Nom, Marque,Modele
FROM Employes E,Autos A
WHERE E.NVoiture = A.NVoiture(+);

/* Jointure droite */
SELECT Nom, Marque,Modele
FROM Employes E,Autos A
WHERE E.NVoiture(+) = A.NVoiture;

/* Jointure complète */
SELECT Nom, Marque,Modele
FROM Employes E,Autos A
WHERE E.NVoiture(+) = A.NVoiture(+);
```

Outre son côté cryptique, cette méthode présente trois inconvénients :
- elle est propre à Oracle et ne peut donc pas être utilisée sur d'autres systèmes ;
- elle place le signe `(+)` à droite dans une jointure gauche et à gauche dans un jointure droite ;
- elle devient inapplicable lorsqu'il y a plus de deux tables.

On considérera donc ce paragraphe comme un tribut à l'histoire.

5.9 Semi-jointure

La notion de semi-jointure recouvre une chose simple. Si par projection, on ne garde que les champs d'une des relations constituant une jointure interne, on obtient une semi-jointure. C'est le cas de la relation suivante, qui ne contient que les employés propriétaires de voiture.

IDEMPLOYE	PRENOM	NOM	DNAISS	DENG	NVOITURE
LC	Laurence	Comhaire	15/03/78	01/09/05	4
JB	Jacques	Bawin	01/02/60	01/09/94	1
RS	Roland	Schyns	24/12/68	05/12/01	3

Dans notre exemple, nous aurions pu parvenir au même résultat en faisant une sélection qui aurait éliminé les lignes dont le champ *NVoiture* contenait la valeur Null. Cette semi-joinure donne cependant une information un peu différente. Ce n'est pas la liste des employés qui possèdent une voiture, mais la liste des employés dont la voiture figure dans la relation *Autos*.

Traduction SQL :

```
SELECT Employes.*
FROM Employes INNER JOIN Ventes
ON Employes.NVoiture = Autos.NVoiture;
```

[4]Encore des beaux exercices en perspective.