

BASE DE DONNEES RELATIONNELLE

2 : Select multi-tables

[Manuels de référence du SQL :](#)

Voir après le sommaire

Bertrand Liaudet

SOMMAIRE

SOMMAIRE	1
Manuels de référence du SQL	2
SQL : INTERROGATION DE LA BD – MULTI-TABLES	3
0. Les opérations s'appliquant à plusieurs tables	3
1. Le modèle relationnel : notion de clé étrangère	4
2. Les jointures naturelles	11
3. Les jointures artificielles	24
4. Les requêtes imbriquées	25
5. Opérations ensemblistes : tables ayant les mêmes attributs	31
6. Equivalence entre les opérations	33
7. Vue	40
8. Produits cartésiens et jointures en SQL 2	43
9. Jointures externes	46
10. CONNECT BY PRIOR : gestion des arborescences	53
ELEMENTS THEORIQUES DE L'ALGEBRE RELATIONNELLE	57
1. Modèle relationnel	57
2. Algèbre relationnelle	60
SGBD – SQL - TP : SELECT MULTI-TABLES	66
Présentation et préparation du travail	66
EXERCICE 1 : charger les tables de la base de données - EmployesTP02.sql	66
EXERCICE 2 : Jointures naturelles - Chapitres 1 et 2 du cours	67
EXERCICE 3 : Jointures artificielles et requêtes imbriquées - Chapitres 3 à 7 du cours	69
EXERCICE 4 : Jointures externes et SQL2 - Chapitres 8 et 9 du cours	70

Manuels de référence du SQL

Chaque SGBD propose son manuel de référence en ligne sur internet.

Mémo SQL : http://www.volubis.fr/bonus/SQL_memo.htm

MySQL

Site officiel MySQL : <http://www-fr.mysql.com/>

Documentations MySQL : <http://dev.mysql.com/doc/index.html>

Documentations MySQL 5.0 en français :

<http://bliaudet.free.fr/IMG/pdf/MySQL-refman-5.0-fr.pdf>

C'est la bible ! On fait les recherches dans le « Search manuel ».

Quand on passe plusieurs mots (logique de « et »), il faut mettre des « + » devant chaque mot.

Pour chercher une phrase, il faut la mettre entre guillemets.

MariaDB

MariaDB est la suite opensource de MySQL

Site officiel MariaDB : <https://mariadb.org/>

ORACLE

http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/toc.htm

PostgreSQL

<http://www.postgresql.org/docs/9.0/static/index.html>

SQL Server

<http://technet.microsoft.com/fr-fr/library/ms173372%28SQL.90%29.aspx>

SQL : INTERROGATION DE LA BD – MULTI-TABLES

0. Les opérations s'appliquant à plusieurs tables

Classification générale

Il y a trois sortes d'opérations s'appliquant à plusieurs tables :

- les jointures
- les opérations ensemblistes
- les requêtes imbriquées et l'utilisation de « vue »

Les jointures se divisent en 3 cas :

- Les jointures naturelles
- Les jointures artificielles
- Les jointures externes apparues avec la norme SQL 2.

Sommaire

Pour traiter d'opérations multi-tables, il faut d'abord aborder la modélisation multi-tables et donc la notion de clé étrangère. Ce sera la première partie :

- 1) Modélisation relationnelle : notion de clé étrangère

Ensuite, le cours suivra la classification des opérations et aura 4 parties :

- 2) Les jointures naturelles en SQL1 et SQL2
- 3) Les jointures artificielles
- 4) Les requêtes imbriquées
- 5) Les opérations ensemblistes

Pour finir on abordera :

- 6) L'équivalence entre les opérations présentées
- 7) La notion de « vue »
- 8) La norme SQL2
- 9) Les jointures externes.

Spécifiquement aux SGBD Oracle et SQL Server, le cours présentera la gestion des arborescences en SQL avec le CONNECT By PRIOR.

1. Le modèle relationnel : notion de clé étrangère

Présentation

Le problème

Au problème précédent de gestion des ressources humaines, on ajoute les spécifications suivantes : Le service du personnel souhaite aussi connaître le nom du département dans lequel l'employé travaille. L'entreprise est répartie dans plusieurs villes. Les départements sont donc caractérisés par leur nom et par leur ville. Un employé travaille dans un département et un seul. Il peut y avoir plusieurs départements qui ont le même nom.

Solution 1 : la mauvaise !

On ajoute les attributs Nom du département et Ville du département dans la table des employés.

RELATION

7 attributs :

Employés	<u>NE</u>	Nom	Fonction	Date d'entrée	Salaire	Comm.	Num. Dept	Nom	Ville
4 tuples :	1	TURNER	SALESMAN	8-SEP-81	3000	0	10	ACCOUNTING	NEW YORK
	2	JAMES	CLERK	3-DEC-81	1800	NULL	30	SALES	CHICAGO
	3	WARD	SALESMAN	22-FEB-81	2500	500	20	RESEARCH	DALLAS
	4	TURNER	ANALYST	3-DEC-81	5000	NULL	10	ACCOUNTING	NEW YORK

Cette solution pose deux problèmes :

- 1) Elle ne permet pas d'avoir des départements vides.
- 2) Elle risque de conduire à des incohérences dans la base si on modifie la ville d'un département. En effet, il faudra modifier tous les tuples concernés pour éviter l'incohérence. Sinon, on se retrouve, comme dans l'exemple, avec un département 10 dans 2 villes différentes.

RELATION

7 attributs :

Employés	<u>NE</u>	Nom	Fonction	Date d'entrée	Salaire	Comm.	Num. Dept	Nom	Ville
4 tuples :	1	TURNER	SALESMAN	8-SEP-81	3000	0	10	ACCOUNTING	DETROIT
	2	JAMES	CLERK	3-DEC-81	1800	NULL	30	SALES	CHICAGO
	3	WARD	SALESMAN	22-FEB-81	2500	500	20	RESEARCH	DALLAS
	4	TURNER	ANALYST	3-DEC-81	5000	NULL	10	ACCOUNTING	NEW YORK

INCOHERENCE !!!

Si dans le tuple numéro 1, on modifie la ville, remplaçant NEW YORK par DETROIT, on aura une BD incohérente puisque dans le tuple 1, le département 10 est à DETROIT et dans le tuple 4, ce même département est à NEW YORK

Solution 2 : la bonne !

On retrouve la table des employés :

RELATION

7 attributs :

Employés	<u>NE</u>	Nom	Fonction	Date d'entrée	Salaire	Comm.	Num. Dept
4 tuples :	1	TURNER	SALESMAN	8-SEP-81	3000	0	10
	2	JAMES	CLERK	3-DEC-81	1800	NULL	30
	3	WARD	SALESMAN	22-FEB-81	2500	500	20
	4	TURNER	ANALYST	3-DEC-81	5000	NULL	10

On va ajouter une table : la table des départements :

RELATION

3 attributs :

Départements	ND	Nom	Ville
4 tuples :	10	ACCOUNTING	NEW YORK
	20	RESEARCH	DALLAS
	30	SALES	CHICAGO
	40	OPERATIONS	BOSTON

- ND est le numéro de département : il était déjà dans la table des employés.
- ND est clé primaire dans la table des départements.
- Désormais, le numéro de département ND de la table des employés fait référence au numéro de département de la table des départements.
- ND est ainsi clé étrangère dans la table des employés.

Avantages

- 1) On a un département vide : le 40
- 2) Si le département 10 de NEW YORK passe à WHASHINGTON, il suffit de modifier l'unique tuple concerné :

RELATION

3 attributs :

Départements	ND	Nom	Ville
4 tuples :	10	ACCOUNTING	WASHINGTON
	20	RESEARCH	DALLAS
	30	SALES	CHICAGO
	40	OPERATIONS	BOSTON

Unique modification : on évite ainsi les incohérences.

Définition

Une clé étrangère est un attribut qui fait référence à une clé primaire.

Schéma de la BD

Le schéma de la BD consiste à écrire chaque table sur une ligne avec les noms de code des attributs :

EMP(**NE**, nom, fonction, dateEmb, sal, comm., #ND)

DEPT (**ND**, nom, ville)

Formalisme usuel pour la clé étrangère

- Le nom d'une clé étrangère est en général le **nom de la clé primaire** qu'elle référence.
- Les clés étrangères sont mises **en dernier dans la liste** des attributs.

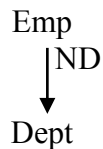
Le nom des clés étrangères est **précédé d'un « # »** sur le papier, mais pas dans la programmation !

Graphe des tables

Le graphe des tables montre les tables et le lien entre les tables :



On peut aussi présenter le nom de l'attribut commun :



Expression synthétique

De façon synthétique, les tables EMP et DEPT peuvent s'écrire ainsi :

E (E, Ex, #D)

D (D, Dx)

On note essentiellement les clés primaires et étrangères. On ne détaille pas les attributs.

Cela permet de se focaliser sur les liens entre les tables, ce qui fait « l'intelligence » du modèle.

C'est utile en phase de réflexion.

Formalisme :

- E, c'est le nom de la table Emp (initial de Emp).
 - E : c'est la clé primaire de la table E
 - Ex : ce sont les attributs de la table E
 - #D : c'est une clé étrangère qui ramène sur la clé primaire D
-
- D, c'est le nom de la table Dept (initial de Dept)
 - D : c'est la clé primaire de la table D
 - Dx : ce sont les attributs de la table D

Présentation

Au problème précédent de gestion des ressources humaines, on ajoute les spécifications suivantes :

Chaque membre du personnel a un supérieur hiérarchique et un seul lui-même membre du personnel, sauf le président qui n'a pas de supérieur hiérarchique.

La solution :

La table résultat ressemblera à celle-ci :

RELATION

8 attributs :

Employés	<u>NE</u>	Nom	Fonction	Date d'entrée	Salaire	Comm.	# ND	*NEchef
5 tuples :	1	TURNER	SALESMAN	8-SEP-81	3000	0	10	5
	2	JAMES	CLERK	3-DEC-81	1800	NULL	30	1
	3	WARD	SALESMAN	22-FEB-81	2500	500	20	4
	4	TURNER	ANALYST	3-DEC-81	5000	NULL	10	5
	5	BOSS	PRESIDENT	10-DEC-80	7000	NULL	5	NULL

Définition

Une clé étrangère réflexive est un attribut qui fait référence la clé primaire de sa table.

Schéma de la BD

Le schéma de la BD consiste à écrire chaque table sur une ligne avec les noms de code des attributs :

EMPLOYES(NE, nom, job, datemb, sal, comm., #ND, *NEchef)

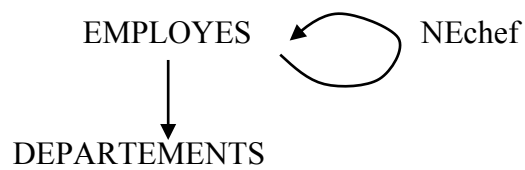
DEPARTEMENTS(ND, nom, ville)

Formalisme complet

1. Les clés **primaires** sont soulignées et placées en premier dans la liste des attributs.
2. Le **nom de la clé primaire** est constitué de : « N »+1^{ère} lettre de la table (NE).
3. Les clés **étrangères** sont précédés d'un #.
4. Les clés **étrangères** sont mises en dernier dans la liste des attributs.
5. Les clés étrangères réflexives sont précédés d'un *.
6. Les clés **étrangères réflexives** sont mises après les clés étrangères non-réflexives.
7. Le **nom des clés étrangères réflexives** est constitué de : clé primaire + code de l'attribut (NEchef)

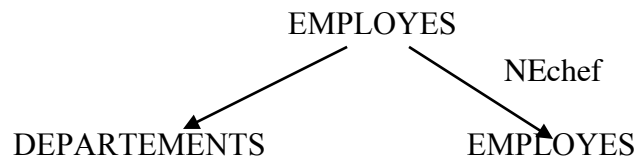
Graphe des tables

Le graphe des tables montre les tables et le lien entre les tables :



On précise le nom de l'attribut clé étrangère réflexive.

On peut aussi dupliquer la table des EMPLOYES :



Définition d'une BD

Une BD c'est un ensemble de tables avec leurs tuples.

Un SGBD gère une ou plusieurs BD.

2. Les jointures naturelles

Exemple

Le modèle

Soit nos 2 tables simplifiées :

➤ **Employés (NE, nom, ND)**

Avec 3 tuples :

NE	Nom	ND
1	Toto	10
2	Tata	10
3	Titi	20

➤ **Départements (ND, ville)**

Avec 2 tuples :

ND	Ville
10	Paris
20	Meudon

La question

On veut afficher les employés avec la ville où ils travaillent.

La solution

```
SELECT *  
FROM Employes, Departement  
WHERE Empolyes.ND = Departement.ND
```

L'explication

➤ **Produit cartésien : FROM Employes, Départements**

Le FROM avec les deux tables est un produit cartésien des deux tables

Employés			X	Département		=	Le produit cartésien des 2 tables				
NE	Nom	ND		ND	Ville	=	NE	Nom	ND	ND	Ville
1	Toto	10		10	Paris		1	Toto	10	10	Paris
2	Tata	10		20	Meudon		1	Toto	10	20	Meudon
3	Titi	20					2	Tata	10	10	Paris
							2	Tata	10	20	Meudon
							3	Titi	20	10	Paris
							3	Titi	20	20	Meudon

➤ **Restriction de jointure : WHERE Empolyes.ND = DepartementND**

Le produit cartésien a permis de réunir tous les attributs : (NE, Nom, ND) + (ND, Ville)

Mais il y a trop de tuples

Le WHERE va permettre de ne garder que les bon tuples : ceux pour lesquels les ND coïncident.

WHERE Empolyes.ND = Departement.ND :

NE	Nom	ND	ND	Ville
1	Toto	10	10	Paris
1	Toto	10	20	Meudon
2	Tata	10	10	Paris
2	Tata	10	20	Meudon
3	Titi	20	10	Paris
3	Titi	20	20	Meudon

➤ **Résultat**

NE	Nom	ND	ND	Ville
1	Toto	10	10	Paris
2	Tata	10	10	Paris
3	Titi	20	20	Meudon

On retrouve nos 3 employés et ils ont toute l'information des département les concernant en plus (ici, uniquement la ville).

Bilan : jointure = PC + restriction de jointure
--

Une jointure est constituée de :

- Un produit cartésien
- Une restriction de jointure

Théorie du Produit cartésien

Définition

Le produit cartésien de deux tables est l'opération de base de la jointure.

Le produit cartésien est l'opération qui va permettre de réunir les attributs de deux tables dans une nouvelle table.

Il produit une table constituée de la concaténation de tous les attributs des deux tables de départ et de tous les tuples formés par la concaténation de chaque tuple de la première table à tous ceux de la deuxième.

Syntaxe SQL

```
Select * from table_1, table_2 ;
```

Exemple

NE	Nom	ND	X	ND	Ville	=	NE	Nom	ND	ND	Ville
1	Toto	10		10	Paris	=	1	Toto	10	10	Paris
2	Tata	10		20	Meudon		1	Toto	10	20	Meudon
3	Titi	20					2	Tata	10	10	Paris
							2	Tata	10	20	Meudon
							3	Titi	20	10	Paris
							3	Titi	20	20	Meudon

Clé primaire

La clé primaire est constituée de la concaténation des clés primaires des tables du produit cartésien.

CP = CP1, CP2

Nombre de tuples

Si la première table contient T1 tuples et la deuxième T2 tuples, le produit cartésien est une table qui contient :

$N1 * N2$ tuples

Nombre d'attributs

Si la première table contient A1 attributs et la deuxième A2 attributs, le produit cartésien est une table qui contient :

$A1 + A2$ attributs.

Usage du produit cartésien

Fondamentalement, le produit cartésien est l'opération qui permet de :

réunir les attributs de deux tables dans une nouvelle table.

On utilise le produit cartésien dès qu'une requête met en jeu deux attributs qui appartiennent à deux tables différentes.

Signification

Le produit cartésien de deux tables n'a jamais de signification.

Il devient significatif quand on l'associe à une restriction qui met en jeu un attribut de la première table et un attribut de la seconde : on l'appelle alors jointure.

Théorie de la jointure

Définition

Jointure = **concept central de l'algèbre relationnelle** et donc des bases de données relationnelles en général.

Comprendre la jointure = comprendre les base de données relationnelle.

La jointure de deux tables c'est leur produit cartésien et la restriction consistant à comparer un attribut de la première table à un attribut de la deuxième table.

Joinure = Produit cartésien + Restriction de jointure

Restriction de jointure = restriction mettant en jeu deux tables.

Syntaxe SQL

```
SELECT *  
FROM table_1, table_2  
WHERE table_1.attribut_1 opérateur table_2.attribut_2;
```

La clause where d'une jointure peut faire intervenir n'importe quel opérateur de l'algèbre booléen. En général, la jointure utilise l'opérateur "=".

Les 2 espèces de jointure

- La jointure naturelle
- La jointure artificielle

La jointure naturelle

Définition

La jointure naturelle est la jointure qui permet de regrouper deux tables ayant un attribut en commun particulier :

- celui de la première table est une clé étrangère,
- celui de la deuxième est la clé primaire pour sa table, clé correspondant à la clé étrangère de la première table.

La comparaison entre les deux attributs est une égalité : opérateur "="

La première table est appelée : « table maître »

La deuxième table est appelée : « table jointe »

Syntaxe SQL

```
SELECT *  
FROM maitre, jointe  
WHERE maitre.clé_étrangère = joint.clé_primaire ;
```

ou encore de façon synthétique, avec c# pour la clé étrangère, cp pour la clé primaire :

```
SELECT *  
FROM m, j  
WHERE m.c# = j.cp;
```

2 types de restriction

➤ *Restriction de jointure naturelle*

La restriction : « m.c# = j.cp » est appelée “restriction de jointure naturelle”.

➤ *Restriction spécifique*

Les restrictions spécifiques sont les restrictions qui ne sont pas des restrictions de jointures.
Ce sont les restrictions propres à la question.

➤ *Exemple*

Les vendeurs et le lieu et le numéro de département où ils travaillent

```
SELECT NE, E.nom, E.ND, ville
FROM employes E, departements D
WHERE E.ND = D.ND           // restriction de jointure naturelle
AND E.fonction = 'SALESMAN' ; // restriction spécifique
```

➤ *Bon usage*

Mieux vaut mettre les restrictions de jointure en premier et les restrictions spécifiques ensuite.

Nombre de tuples d'une jointure naturelle

Si la table maître contient TM tuples, la jointure naturelle est une table qui contient :
Entre 0 et TM tuples

On perd des tuples si la clé étrangère de la table maître vaut NULL : dans ce cas elle ne sera jamais égal à la clé primaire de la table jointe et le tuple est perdu.

Si la clé étrangère de la jointure est « not NULL », il n'y a pas de restriction dans la table maître et la jointure naturelle est une table qui contient :
TM tuples

Nombre d'attributs

Même règle que pour le produit cartésien : on retrouve les tuples des deux tables.

Signification : sur-informer une table

La jointure naturelle permet de "sur-informer" la clé étrangère de la première table : c'est-à-dire de faire entrer dans la première table tous les attributs correspondant à la clé étrangère et qu'on trouve dans la deuxième table.

Clé primaire

La clé primaire d'une jointure naturelle est constituée par la clé primaire de la table maître.
CP = CP_TM

Graphe de la question

Le graphe de la question représente les tables et les attributs en jeu pour une requête donnée. On l'abordera dans les exemples.

Exemple 1

La jointure entre la table employes et la table departements s'écrit :

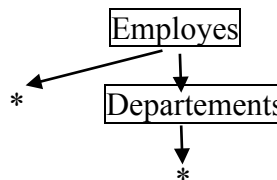
```
SELECT *  
FROM employes E, departements D  
WHERE E.ND = D.ND ;
```

La table résultant de la jointure contient autant de tuples que la table employés : ce sont tous les employés auxquels on a concaténé le tuple de la table départements qui leur correspond.

➤ Remarque

Dans la restriction de jointure, on précise la table d'origine de l'attribut (ND) car dans la table résultant du produit cartésien de employes et departements, il y a deux fois l'attribut ND. Pour préciser duquel on parle, on précise sa table d'origine.

➤ Graphe de la question



Les tables sont encadrées.

Les flèches montrent les attributs projetés. * signifie : tous les attributs.

La flèche entre deux tables est un lien de jointure naturelle : elle signifie la restriction de jointure naturelle.

Exemple 2

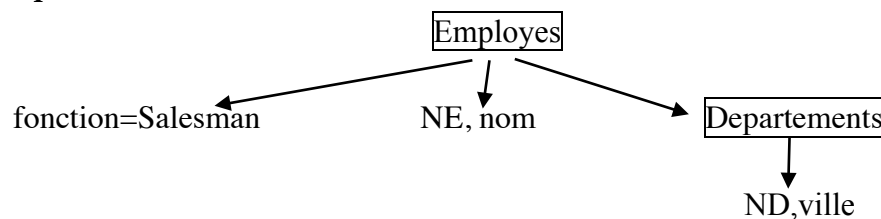
Tous les employés appelés SALESMAN et le lieu et le numéro de département où ils travaillent

```
SELECT NE, E.nom, E.ND, ville
FROM employes E, departements D
WHERE E.ND = D.ND
AND E.fonction = 'SALESMAN' ;
```

➤ Résultats

NE	nom	ND	ville
7840	MARTIN	30	CHICAGO
7841	ALLEN	30	CHICAGO
7842	TURNER	30	CHICAGO
7844	WARD	30	CHICAGO

➤ Graphe de la question



Les tables sont encadrées.

Les flèches montrent les attributs projetés.

Fonction = Salesman signifie qu'il y a une restriction.

La flèche entre deux table est un lien de jointure naturelle : elle signifie la restriction de jointure naturelle.

Remarque 1

Comme pour toute restriction-projection primaire, **on projette, pour éviter les doublons, la clé primaire de la table résultant de la jointure**, c'est-à-dire la clé primaire de la table maitre.

Remarque 2

Il faut préfixer tous les attributs qu'on retrouve dans les deux tables : ND, mais aussi « nom » qui se trouve dans les deux tables : le nom des employés et le nom des départements.

Remarque 3

On aurait pu tout aussi bien choisir départements.ND plutôt que employésND puisque la condition de jointure n'a conservé que des valeurs identiques pour ces deux attributs.

Jointure naturelle réflexive : jointure d'une table avec elle-même

Définition

Les jointures naturelles réflexives concernent les tables qui contiennent une clé étrangère qui fait référence à la clé primaire de la même table.

Syntaxe

Dans ce cas, il va falloir renommer les tables pour pouvoir les distinguer :

```
SELECT *  
FROM table_1 synonyme1, table_2 synonyme2  
WHERE synonyme1.attribut_1 = synonyme2.attribut_2;
```

Exemple

Tous les employés du département 20 et leurs supérieurs hiérarchiques :

```
SELECT E.NE, E.nom, E.ND, Echef.NE, Echef.nom  
FROM employees E, employees Echef  
WHERE E.NEchef = Echef.NE  
AND E.ND = 20;
```

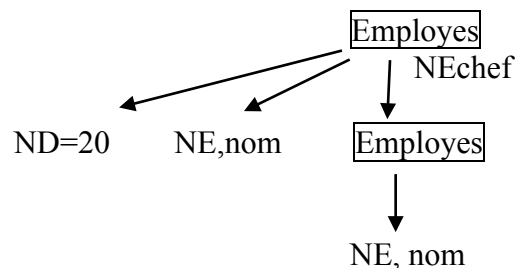
➤ Résultats

NE	nom	ND	NE	nom
7566	JONES	20	7839	KING
7845	FORD	20	7566	JONES
7846	SMITH	20	7845	FORD
7847	SCOTT	20	7566	JONES
7848	ADAMS	20	7847	SCOTT

Remarque

Dans le cas de la jointure de deux tables identiques, tous les attributs de la table résultant du produit cartésien sont en double. Il faut donc forcément préciser pour chaque attribut de projection sa table d'origine.

Graphe de la question



Les tables sont encadrées : on a 2 tables d'employés.

Les flèches montrent les attributs projetés.

La flèche entre deux tables est un lien de jointure naturelle : elle signifie la restriction de jointure naturelle. On précise le nom de la clé étrangère réflexive.

ND=20 signifie qu'il y a une restriction.

Jointure de plus de deux tables

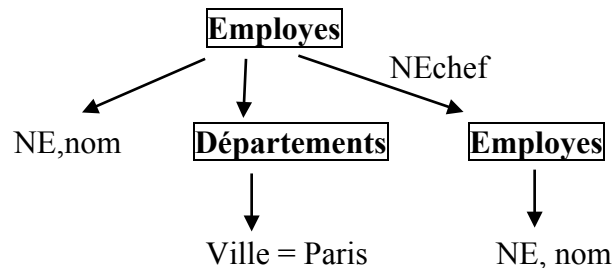
On peut faire des jointures sur autant de tables que l'on veut.
Il y a alors autant de restrictions qu'il y a de tables moins une.

Exemple

Tous les employés du parisiens et leurs supérieurs hiérarchiques :

```
SELECT E.NE, E.nom, D.ville, Echef.NE, Echef.nom  
FROM employes E, employes Echef, departements D  
WHERE E.NEchef = Echef.NE  
AND E.ND = D.ND  
AND D.ville = 'Paris' ;
```

Graphe de la question



Les tables sont encadrées : on a 2 tables d'employés et une table de départements.

Les flèches montrent les attributs projetés.

La flèche entre deux tables est un lien de jointure naturelle : elle signifie la restriction de jointure naturelle. On précise le nom de la clé étrangère réflexive.

Ville = Paris signifie qu'il y a une restriction.

Jointure et ONLY FULL GROUP BY

Présentation

On veut récupérer le numéro et toutes les caractéristiques des départements avec le nombre d'employés par département.

➤ *Solution standard SQL : ONLY FULL GROUP BY*

```
SELECT D.ND, D.nom, D.ville, count(*) as NbEmployes
FROM employes E, departments D
WHERE E.ND = D.ND
GROUP BY D.ND, D.nom, D.ville
```

On est obligé de mettre tous les attributs du département dans le GROUP BY pour pouvoir les projeter.

➤ *Solution MySQL non standard SQL : sans ONLY FULL GROUP BY*

```
SELECT D.ND, D.nom, D.ville, count(*) as NbEmployes
FROM employes E, departments D
WHERE E.ND = D.ND
GROUP BY D.ND
```

On ne met que ND dans le GROUP BY. C'est suffisant pour faire le count(*). L'absence d'ONLY FULL GROUP BY permet de projeter n'importe quel attribut en plus.

Cette technique n'étant pas standard, on l'évitera systématiquement pour prendre de bonnes habitudes !

➤ *Erreur non corrigée sans ONLY FULL GROUP BY*

Sans ONLY FULL GROUP BY, on peut projeter n'importe quel attribut autre que ceux du GB. Ainsi cette requête, qui ne veut rien dire, va pourtant s'exécuter :

```
SELECT D.ND, E.nom, D.ville, count(*) as NbEmployes
FROM employes E, departments D
WHERE E.ND = D.ND
GROUP BY D.ND
```

ONLY FULL GROUP BY de MySQL

➤ *Variable système SQL_MODE et « ONLY_FULL_GROUP_BY »*

La variable « sql_mode » permet de paramétrer un « group by » standard.
Pour cela, elle doit prendre la valeur : ONLY_FULL_GROUP_BY

➤ *Consultation de la variable système SQL_MODE*

```
SHOW variables like '%SQL%'
```

Par défaut, à l'installation, la variable « sql_mode » contient :
STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
Soit 3 caractéristiques différentes. Cette situation peut changer selon les versions de MySQL.

➤ *Modification de la variable système SQL_MODE*

Pour modifier une variable système VS, on utilise la commande SET.
On peut modifier une VS au niveau local (pour le client seulement) ou au niveau global (au niveau du serveur, pour tous les clients).
A noter qu'en redémarrant le serveur, on revient à la version du SQL_MODE qui est dans le fichier de configuration.

Pour supprimer le contenu de SQL_MODE :

```
SET GLOBAL SQL_MODE = '';
```

Pour donner un contenu unique à SQL_MODE :

```
SET GLOBAL SQL_MODE = 'ONLY_FULL_GROUP_BY';
```

Pour ajouter une nouvelle caractéristique à SQL_MODE :

```
SET GLOBAL SQL_MODE = concat(@@sql_mode,',ONLY_FULL_GROUP_BY');
```

On peut aussi écrire :

```
set @@GLOBAL.SQL_MODE = concat(@@sql_mode,',ONLY_FULL_GROUP_BY');
```

Pour que l'action ne soit que pour le client :

```
set @@LOCAL.SQL_MODE = concat(@@sql_mode,',ONLY_FULL_GROUP_BY');
```

Ou encore

```
SET LOCAL SQL_MODE = concat(@@sql_mode,',ONLY_FULL_GROUP_BY');
```

➤ *Modification de la variable système SQL_MODE dans le fichier de configuration*

On peut aussi modifier la valeur du sql_mode dans le fichier de configuration : my.ini :

```
[mysqld]  
sql_mode=  
STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION,  
ONLY_FULL_GROUP_BY'
```

Jointure naturelle et SQL2

Présentation

Le SQL 2 introduit une nouvelle syntaxe pour les jointures.

Cette syntaxe permet de distinguer entre les restrictions de jointures et les restrictions spécifiques.

On présente ici un premier usage du SQL2 pour la jointure naturelle à partir d'exemples.

On présentera toute la syntaxe SQL2 dans la suite du cours.

Exemples

➤ 1 : principe général

SQL1 :

```
SELECT *  
FROM maitre, jointe  
WHERE maitre.clé_étrangère = jointe.clé_primaire ;
```

SQL2 :

```
SELECT *  
FROM maitre  
JOIN jointe ON maitre.clé_étrangère = jointe.clé_primaire ;
```

Le JOIN remplace la virgule du produit cartésien.

La restriction de jointure est mise dans un ON

On met sur une même ligne le JOIN et le ON

➤ 2 : les employés et leurs départements

SQL1 :

```
SELECT *  
FROM Employes, Departements  
WHERE Employes.ND = Departements.ND;
```

SQL2 :

```
SELECT *  
FROM Employes  
JOIN Departements ON Employes.ND = Departements.ND;
```

➤ 3 : les vendeurs et le lieu et le numéro de département où ils travaillent

SQL1 :

```
SELECT NE, E.nom, E.ND, ville  
FROM employes E, departements D  
WHERE E.ND = D.ND // restriction de jointure naturelle  
AND E.fonction = 'SALESMAN' ; // restriction spécifique
```

SQL2:

```
SELECT NE, E.nom, E.ND, ville  
FROM employes E  
JOIN departements D ON E.ND = D.ND // restrict. de join. Nat.  
WHERE E.fonction = 'SALESMAN' ; // restriction spécifique
```

➤ **4 : les employés du département 20 et leurs supérieurs hiérarchiques**

SQL1 :

```
SELECT E.NE, E.nom, E.ND, Echef.NE, Echef.nom
FROM employes E, employes Echef
WHERE E.NEchef = Echef.NE
AND E.ND = 20;
```

SQL2 :

```
SELECT E.NE, E.nom, E.ND, Echef.NE, Echef.nom
FROM employes E
JOIN employes Echef ON E.NEchef = Echef.NE
WHERE E.ND = 20;
```

➤ **5 : Tous les employés de Dallas et leurs supérieurs hiérarchiques :**

SQL1 :

```
SELECT E.NE, E.nom, D.ville, Echef.NE, Echef.nom
FROM employes E, employes Echef, departements D
WHERE E.NEchef = Echef.NE
AND E.ND = D.ND
AND D.ville='Dallas';
```

SQL2 :

```
SELECT E.NE, E.nom, D.ville, Echef.NE, Echef.nom
FROM employes E
JOIN employes Echef ON E.NEchef = Echef.NE
JOIN departements D ON E.ND = D.ND
WHERE D.ville='Dallas';
```

C'est avec ce dernier exemple qu'on peut voir que la syntaxe SQL2 est plus claire que celle du SQL1 : on fait les JOIN-ON un par un, puis les restrictions spécifiques.

A noter qu'on peut aussi écrire :

```
SELECT E.NE, E.nom, D.ville, Echef.NE, Echef.nom
FROM employes E
JOIN employes Echef ON E.NEchef = Echef.NE
JOIN departements D ON E.ND = D.ND AND D.ville='Dallas';
```

Le ON peut donc servir pour la restriction de jointure mais aussi pour les restrictions spécifiques.

3. Les jointures artificielles

Définition

Jointure en général

La jointure de deux tables c'est leur **produit cartésien** et la **restriction** consistant à comparer un attribut de la première table à un attribut de la deuxième table.

```
Select *  
from table_1, table_2  
where table_1.attribut_1 opérateur table_2.attribut_2;
```

Jointure naturelle

Dans une jointure naturelle, la restriction concerne un attribut clé primaire et un attribut clé étrangère y faisant référence.

La comparaison entre les deux attributs est une égalité : opérateur "="

Cette restriction peut être appelée **restriction de jointure naturelle**.

Jointure artificielle

La jointure artificielle est une jointure dont la restriction n'est **PAS une restriction de jointure naturelle** : soit elle met en jeu d'autres opérateurs que celui d'égalité, soit elle met en jeu un couple d'attribut qui n'est pas clé-étrangère et clé-primaire correspondante.

Exemple

Tous les employés ayant le même job que JONES :

```
SELECT DISTINCT e1.NE, e1.nom, e2.job  
FROM emp e1, emp e2  
WHERE e1.job = e2.job      -- restriction de jointure artificielle  
AND e2.ename = 'JONES'    -- restriction spécifique 1  
AND e1.ename <> 'JONES'    -- restriction spécifique 2
```

Explications

Chaque employé est croisé avec tous les employés (produit cartésien).

On ne s'intéresse qu'aux employés de e2 qui s'appelle JONES (restriction spécifique 1)

La restriction de jointure artificielle est faite sur job : on ne garde que les employés de e1 qui ont le même job que les JONES.

On supprime JONES de la liste des employés de e1 (pour éviter d'avoir JONES dans la réponse).

Pour finir, on projette les employés de e1.

On met un distinct pour le cas où il y ait plusieurs JONES.

4. Les requêtes imbriquées

Présentation

Généralités

Dans une requête imbriquée, on trouve un ou plusieurs SELECT utilisés dans le SELECT PRINCIPAL (le premier de la requête).

On distingue entre le select imbriquant (ou principal) et le ou les select imbriqués.

On peut avoir autant de niveau d'imbrication qu'on veut.

3 types de select imbriqués

- Les **SELECT imbriqués dans le SELECT** : le select imbriqué remplace un attribut.
- Les **SELECT imbriqués dans le FROM** : le select imbriqué remplace une table.
- Les **SELECT imbriqués dans le WHERE** : le select remplace une valeur ou une liste de valeurs possibles dans une restriction.

2 types de SELECT imbriqués dans le WHERE

Deux types de SELECT imbriqué imbriqué dans le WHERE selon la restriction effectuée :

- Comparaison de chaque tuple du select imbriqué aux tuples du select principal : ALL et ANY.
- Test de la cardinalité vide ou pas de la table résultat du select imbriqué : EXISTS et NOT EXISTS.

SELECT imbriqué dans le SELECT

Si un select renvoie une seule ligne, il peut être considéré comme un attribut

➤ *Exemple : moyenne des salaires par n° de département et moyenne générale des salaires*

```
SELECT nd, avg(sal), (SELECT avg(sal) FROM emp) moy_gen
FROM emp
GROUP BY nd
;
```

➤ *Remarques syntaxiques*

- Le select imbriqué est mis entre parenthèses.
- Le select imbriqué doit ne renvoyer qu'une seule ligne : en général, c'est une fonction de groupe.
- Le select imbriqué est renommé.

SELECT imbriqué dans le FROM

Toute table d'un select peut être remplacée par un autre select.

➤ *Exemple : nombre moyen d'employés par département*

```
SELECT avg(nb)
FROM (
    SELECT count(*) nb
    FROM emp
    GROUP BY nd ) tnb
;
```

➤ *Remarques syntaxiques*

- Le select imbriqué est mis entre parenthèses.
- Le select imbriqué est obligatoirement renommé (« tnb » dans notre exemple).
- Les attributs projetés dans le select imbriqué sont obligatoirement renommés quand il s'agit de fonction de groupe.

SELECT imbriqué dans le WHERE : ALL et ANY

Présentation

Le select imbrique dans le where avec opérateur ALL ou ANY consiste en une comparaison de chaque tuple du select principal avec les tuples du select imbriqué.

Opérateurs ANY et IN

- *Exemple : tous les employés ayant le même job que les employés du département 10*

```
SELECT NE, nom, job FROM emp
WHERE job =ANY (
    SELECT job FROM emp
    WHERE ND = 10
);
```

= any : = n'importe lequel dans la table résultat du SELECT imbriquée

La comparaison booléenne est vraie si elle est vraie pour au moins un tuple du select imbriqué (le deuxième select).

Autrement dit, « = any » vérifie l'appartenance à la liste des tuples du deuxième select.

- *On peut écrire IN à la place de =ANY*

```
SELECT NE, nom, job FROM emp
WHERE job IN (
    SELECT job FROM emp
    WHERE ND = 10
);
```

Opérateurs ALL et not IN

- *Exemple : tous les employés ne travaillant pas à PARIS*

```
SELECT NE, nom, job FROM emp
WHERE nd != ALL (
    SELECT nd FROM dept
    WHERE VILLE = « PARIS »
);
```

!=ALL : différent de tout ceux de la table résultat.

La comparaison booléenne est vraie si elle est vraie pour tous les tuples du select imbriqué (le deuxième select).

Autrement dit, « != all » vérifie la non-appartenance à la liste des tuples du deuxième select.

- *On peut écrire not IN à la place de !=ALL*

```
SELECT NE, nom, job FROM emp
WHERE nd not IN (
    SELECT nd FROM dept
    WHERE VILLE = « PARIS »
);
```

Syntaxe générale

La syntaxe générale est la suivante :

```
SELECT * from table
WHERE liste 1 d'attributs opérateur
      (SELECT liste 2 d'attributs FROM ... );
```

➤ *L'opérateur*

C'est un opérateur de comparaison booléen classique : =, !=, >, >=, <, <=, suivi d'un opérateur spécial : ANY ou ALL.

➤ *Les listes d'attributs*

Elles doivent avoir la même forme dans le WHERE du select principal et dans la projection du select imbriqué : même nombre d'attributs et même type pour les attributs. Habituellement, ce sont les mêmes attributs dans le select principal et dans le SELECT imbriqué.

➤ *Opérateurs IN et not IN*

L'opérateur « in » remplace « = any ».

L'opérateur « not in » remplace « != all »

➤ *Cas où on peut se passer de ANY et de ALL*

Si le deuxième SELECT fournit un seul tuple, on peut se passer de l'opérateur spécial. On peut par exemple écrire "=" à la place de "=ANY".

C'est le cas particulièrement quand le select imbriqué renvoie une fonction de groupe ou quand une restriction s'est faite sur une valeur de la clé primaire.

SELECT imbriqué dans le WHERE : EXISTS et NOT EXISTS

Présentation

Le select imbrique dans le where avec opérateur EXISTS et NOT EXISTS consiste, **pour chaque tuple du select principal, à tester si la cardinalité du select imbriqué est vide ou pas.**

Opérateur EXISTS

Les tuples du select principal sont sélectionnés si le select imbriqué produit au moins un tuple.

➤ *Exemple : tous les employés travaillant dans un département qui contient au moins un 'ANALYST' (c'est un métier) :*

```
SELECT NE, nom, ND, job FROM emp
WHERE exists (
    SELECT * from emp e1
    WHERE job = 'ANALYST'
    AND e1.ND = emp.ND
);
```

➤ *Explication*

On regarde s'il existe un ANALYST dont le département est égal à celui du tuple en cours. Si oui, on garde le tuple en cours.

➤ *Remarque*

Dans le cas d'un select imbriqué dans le where avec EXISTS, on a forcément une restriction de jointure artificielle dans le select imbriqué, sinon le résultat du select imbriqué serait constant et le test EXISTS donnerait soit toujours vrai, soit toujours faux, donc on obtiendrait finalement soit la conservation de tous les tuples du select imbriquant, soit l'élimination de tous les tuples du select imbriquant.

Opérateur NOT EXISTS

➤ *Exemple : tous les employés travaillant dans un département sans 'ANALYST' (c'est un métier) :*

```
SELECT NE, nom, ND, job FROM emp
WHERE not exists (
    SELECT * from emp e1
    WHERE job = 'ANALYST'
    AND e1.ND = emp.ND
);
```

Les tuples du select principal sont sélectionnés si le select imbriqué ne produit aucun tuple.

Syntaxe

La syntaxe générale est la suivante :

```
SELECT * FROM table
WHERE [not]EXIST (
    SELECT ...
    WHERE table.attribut ...
);
```

[not] veut dire qu'on peut avoir not ou pas.

En général, le select imbriqué utilise un attribut du select principal qui rend variable son résultat en fonction du tuple en cours du select principal.

Equivalence entre IN et EXISTS et entre NOT IN et NOT EXISTS

A la place de l'exemple précédent avec un EXISTS , on pourrait écrire avec un IN:

```
SELECT NE, nom, ND, job FROM emp e
WHERE ND IN (
    SELECT ND FROM emp
    WHERE = 'ANALYST'
);
```

Différences entre le IN et le EXISTS

La différence entre le IN et le EXISTS concerne la gestion des valeurs NULL.

Quand on utilise un IN, si le select imbriqué renvoie une valeur NULL, le IN vaudra toujours FAUX.

5. Opérations ensemblistes : tables ayant les mêmes attributs

Principe général des opérations ensemblistes

Les opérations ensemblistes s'appliquent à deux tables qui ont les mêmes attributs (où plus précisément des attributs de mêmes types).

Il y a trois opérations ensemblistes basique en SQL :

- **UNION** : regrouper tous les tuples de 2 tables
- **INTERSECTION** : extraire les tuples en commun dans 2 tables
- **DIFFERENCE** : prendre les tuples d'une table qui ne sont pas dans l'autre.

Remarque :

Pour que ces opérations soient significatives, il ne suffit pas que les attributs aient le même type, encore faut-il qu'ils aient la même signification. Dans le cas le plus général, ils seront identiques.

UNION

Exemple : les employés avec leur chef en gardant le chef

```
SELECT e.NE, e.nom, echef.NE, echef.nom
FROM emp e, emp echef
WHERE e.NEchef = echef.NE
UNION
SELECT e.NE, e.nom, NULL, NULL
FROM emp e
WHERE e.NEchef is NULL ;
```

La jointure nous fait perdre les chefs dont le NEchef vaut NULL.

On fait un deuxième SELECT avec seulement ceux dont le NEchef vaut NULL

Et on fait l'UNION des deux SELECT.

A noter qu'on a rajouté NULL, NULL pour avoir le même nombre d'attributs.

Syntaxe

L'union de deux tables ayant des attributs de mêmes types est une table constituée des attributs de la première table et de tous les tuples des deux tables de départ. La syntaxe est la suivante :

```
SELECT liste_1 d'attributs FROM table_1
union
SELECT liste_2 d'attributs FROM table_2;
```

➤ *Les listes d'attributs*

Elles doivent avoir la même forme dans le where du select principal et dans la projection du select imbriqué : même nombre d'attributs et même type pour les attributs. Habituellement, ce sont les mêmes attributs dans les deux select.

➤ *Les tables des select*

Les deux tables de départ (table_1 et table_2) peuvent avoir des attributs différents. Ce sont les tables projetées qui doivent avoir des attributs de mêmes types pour pouvoir être unies.

➤ *Suppression des doublons*

Les éventuels doublons sont éliminés automatiquement par l'UNION, comme par toutes les opérations ensemblistes.

➤ *Ordre des tuples*

Les tuples apparaissent dans l'ordre du premier select suivi par ceux du deuxième select.

Toutefois, le premier select ne peut pas contenir de ORDER BY

On peut trier le résultat final en mettant un ORDER BY à la fin du deuxième select.

DIFFERENCE : opérateur MINUS

Exemple

Tous les numéros des départements vides de la société (c'est-à-dire les numéros des départements dans lesquels aucun employé ne travaille) :

```
SELECT deptno FROM dept
MINUS
SELECT DISTINCT deptno FROM emp;
```

On soustrait de la table des numéros de tous les départements la table des numéros des départements non vides (ceux dans lesquels travaille au moins un employé).

Syntaxe

La différence de deux tables ayant des attributs de mêmes types est une table constituée des attributs de la première table et des tuples de la première table n'appartenant pas à la deuxième :

```
SELECT liste_1 d'attributs FROM table_1
MINUS
SELECT liste_2 d'attributs FROM table_2;
```

Intersection

L'intersection de deux tables ayant des attributs de mêmes types est une table constituée des attributs de la première table et des tuples de la première table appartenant aussi à la deuxième :

```
SELECT liste_1 d'attributs FROM table_1
INTERSECT
SELECT liste_2 d'attributs FROM table_2;
```

Remarque MySQL

MySQL ne propose que l'UNION.

Les deux autres opérateurs peuvent toujours être remplacés par d'autres opérateurs de calcul (des SELECT imbriquées ou des jointures).

L'exemple présenté pour l'UNION pourra aussi être remplacé par une jointure externe.

6. Equivalence entre les opérations

Présentation

De nombreux cas

Il existe de nombreuses équivalences entre différentes opérations de l'algèbre relationnelle. On ne va pas les présenter toutes, mais présenter seulement les plus significatives.

Raisons de choisir une méthode plutôt qu'une autre

Il y a deux raisons pour choisir une méthode plutôt qu'une autre :

- **L'optimisation d'exécution** : une méthode s'exécute plus vite que l'autre. On peut définir des préférences théoriques pour une méthode plutôt que pour une autre. Toutefois, on peut aussi vérifier concrètement, en fonction du SGBD, si une telle méthode est effectivement plus efficace que telle autre. La commande EXPLAIN permet ce genre d'analyse.
- **L'optimisation de maintenance** : une méthode est plus facile à maintenir (c'est-à-dire plus facile à comprendre) que l'autre. Cette raison est subjective : ça dépendra des usages de l'entreprise.

Equivalence entre jointure naturelle et select imbriqué dans le where

On peut très souvent transformer jointures naturelles en select imbriqués dans le where.

Exemple : tous les employés travaillant à Paris avec leurs n° et leurs noms

Version avec jointure naturelle :

```
Select e.NE, e.nom,  
from emp e, dept d  
where e.ND = d.ND  
and d.ville = 'PARIS';
```

équivalent à :

Version avec select imbriqué dans le where avec IN

```
Select NE, nom, job from emp  
where ND = any (  
    Select ND from dept  
    where ville = 'PARIS';  
);
```

équivalent à :

Version avec select imbriqué dans le where avec EXISTS

```
Select NE, nom, job from emp e  
where EXISTS (  
    Select ND from dept d  
    where ville = 'PARIS'  
    and e.ND = d.ND  
);
```

Cas où la transformation est impossible

Si on veut projeter des attributs de la table imbriquée (ici : « dept »), il faut forcément passer par une jointure naturelle.

Du bon usage

➤ Optimisation d'exécution

On choisira toujours la jointure naturelle.

➤ Optimisation de maintenance

On choisira toujours la jointure naturelle.

Equivalence entre jointure artificielle et select imbriqué dans le where

On peut très souvent transformer les select imbriqués en jointure artificielle.

Exemple : tous les employés ayant le même job que les employés du département 10

Version avec select imbriqué dans le where avec IN

```
Select NE, nom, job from emp
where job = any (
    Select job from emp
    where ND = 10
)
and ND != 10 ;
```

équivalent à :

Version avec jointure artificielle :

```
Select e1.empno, e1.ename, e1. job
from emp e1, emp e2
where e1.job = e2.job
and e2.ND = 10
and e1.ND != 10 ;
```

équivalent à :

Version avec select imbriqué dans le where avec EXISTS

```
Select NE, nom, job from emp e1
where EXISTS (
    Select job from emp e2
    where ND = 10
    and e1.job = e2.job
)
and ND != 10 ;
```

Remarque

On ajoute « **and** ND != 10 » pour éviter de projeter les employés du département 10 qui ont évidemment le même job que les employés du département 10.

Cas où la transformation est impossible

Si on veut projeter des attributs de la table imbriquée (ici : « dept »), il faut forcément passer par une jointure naturelle.

Quand on a un NOT IN ou un NOT EXISTS, si le select imbriqué renvoie plusieurs tuples, on ne peut pas transformer la requête imbriquée en jointure artificielle. On pourra les transformer en opérations ensemblistes ou en jointures externes.

Du bon usage

➤ *Optimisation d'exécution*

On choisira la jointure artificielle.

➤ *Optimisation de maintenance*

La jointure artificielle étant un peu compliquée à comprendre, l'optimisation de la maintenance peut conduire à préférer le select imbriqué à la jointure artificielle.

La clause EXISTS est un peu compliquée à comprendre. L'optimisation de la maintenance peut donc aussi conduire à préférer la clause IN à la clause EXISTS.

Equivalence entre select imbriqué dans le FROM et select imbriqué dans le WHERE

Quand le select imbriqué porte sur un résultat statistique, on ne peut pas la transformer en jointure. Par contre on peut transformer un select imbriqué dans le where en un select imbriqué dans le from.

Exemple : tous les employés gagnants plus que la moyenne

Version avec select imbriqué dans le where :

```
Select NE, nom, sal
from emp
where sal > (
    Select avg(sal) from emp
);
```

équivalent à :

Version avec select imbriqué dans le from :

```
Select NE, nom, sal, moysal
from emp, (Select avg(sal) as moysal from emp) tmoy
where sal > moysal;
```

Cas où la transformation est impossible

Si on veut projeter des attributs statistiques de la table imbriquée (ici : « avg(sal) »), il faut forcément passer par un select imbriqué dans le from.

Du bon usage

➤ ***Optimisation d'exécution***

Les deux méthodes sont équivalentes théoriquement.

➤ ***Optimisation de maintenance***

Le select imbriqué dans le from est un peu compliquée à comprendre On choisira donc le select imbriqué dans le where de préférence à un select imbriqué dans le from pour des questions d'optimisation de maintenance.

Equivalence entre INTERSECT et SELECT imbriqué dans le WHERE avec IN

Principe

Version avec intersect :

```
Select listeAtt1 from table1  
intersect  
Select listeAtt1 from table2
```

équivalent à :

Version avec select imbriqué dans le where et IN

```
Select listeAtt1 from table1  
Where listeAtt1 in (  
    Select listeAtt1 from table2  
);
```

Du bon usage

➤ *Optimisation d'exécution*

Les deux méthodes sont équivalentes théoriquement.

➤ *Optimisation de maintenance*

L'usage du select imbriqué étant plus courant que celui de l'intersect, et MySQL par exemple ne proposant pas l'opérateur « INTERSECT », on choisira le select imbriqué dans le where de préférence à un « intersect » des questions d'optimisation de maintenance.

Equivalence entre MINUS et select imbriqué dans le where avec NOT IN

tous les numéros des départements vides de la société

On cherche tous les numéros des départements dans lesquels aucun employé ne travaille.

Version avec minus :

```
Select deptno from dept
minus
select distinct deptno from emp;
```

équivalent à :

Version avec select imbriqué dans le where et NOT IN

```
Select ND from dept
Where ND not in (
    select ND from emp
);
```

Remarque

On a vu dans le paragraphe sur les équivalences entre jointure artificielle et select imbriqués que le « NOT IN » ne peut pas être transformé en jointure artificielle

Du bon usage

➤ *Optimisation d'exécution*

Les deux méthodes sont équivalentes théoriquement.

➤ *Optimisation de maintenance*

L'usage du select imbriqué étant plus courant que celui du minus, et MySQL par exemple ne proposant pas l'opérateur « MINUS », on choisira le select imbriqué dans le where de préférence à un « MINUS » des questions d'optimisation de maintenance.

7. Vue

Présentation

- Une commande **select** peut être conservée dans un objet de la BD appelé "vue".
- On peut ensuite utiliser cet objet de type « vue » comme une table. On peut faire des SELECT et, sous certaines conditions, on peut ajouter, modifier ou supprimer des tuples dans une vue et par conséquent dans la ou les tables associées à la vue.
- Le « contenu » d'une vue n'a pas d'existence physique : il est recalculé à chaque utilisation de la vue qui se comporte donc comme un select.
- Une vue peut être considérée comme une fonction sans paramètres formels retournant une table.

Gestion des vues

Create view

La syntaxe de la création d'une vue est la suivante :

```
CREATE VIEW nom_vue AS  
select ...
```

ou

```
CREATE or REPLACE VIEW nom_vue AS  
select ...
```

Drop view

La syntaxe de la suppression d'une vue est la suivante :

```
DROP VIEW nom_vue ;
```

Consultation du code d'une vue

```
mysql> SHOW CREATE VIEW nom_vue;
```

Utilisation d'une vue

Une vue s'utilise comme une table :

```
mysql> SELECT * FROM nom_vue;
```

Les 2 usages des vues

1 : décomposer les requêtes en sous-requêtes

Une vue peut remplacer n'importe quelle table dans un select.

Les vues permettent ainsi de décomposer l'écriture des requêtes complexes.

Le select principal joue le rôle de programme principal. Les vues correspondant à des sous-requêtes qui peuvent prendre la place d'une table ou d'un attribut si la vue ne retourne qu'une ligne.

Exemple

Tous les employés travaillant dans un département qui contient au moins un 'ANALYST' (c'est un métier).

On commence par créer la vue correspondant à la requête : « tous les départements contenant au

moins un ANALYST (cf. exemple précédent) :

```
CREATE or REPLACE VIEW deptAvecAnalyste as
Select distinct nd from emp
where job = 'ANALYST'
order by nd;
```

Ensuite on traite la requête principale en exploitant la nouvelle vue :

```
Select ne, nom from emp
where nd in (select nd from deptAvecAnalyste);
```

ou :

```
Select distinct e.ne, e.nom from emp e, deptAvecAnalyste d
where e.nd =d.nd;
```

2 : Faciliter et sécuriser l'accès aux données par les utilisateurs

On peut créer des vues et ne donner des droits d'accès que sur ces vues : ça favorise la sécurisation et ça facilite l'accès, les vues étant orientées « usage ».

La vue permet aussi de restreindre l'accès aux tuples.

Pour que cet usage soit pertinent, il faut pouvoir appliquer les opérations du DML (INSERT, UPDATE, DELETE) à une vue.

Exemple

On veut limiter les droits d'accès de toto@localhost à la consultation du n°, du nom et du job des employés des départements 10 et 20 (table emp, BD empdept) :

```
Create view vueEmp as
Select ne, nom, job
From emp where nd in (10, 20);
```

```
Grant SELECT on empdept.vueEmp to toto@localhost;
```

Les vues modifiables

INSERT, UPDATE et REPLACE dans une vue

Vue mono-table

On peut faire des INSERT, des UPDATE et des REPLACE dans une vue mono-table : ils passeront dans la table correspondante, à condition que toutes les contraintes d'intégrité soient correctement prises en compte.

Vue multi-table

On peut faire des INSERT, un UPDATE ou un REPLACE dans une vue multi-table à condition que toutes les contraintes d'intégrité de la table correspondante soient correctement prises en compte. Mais on ne peut créer, modifier ou remplacer qu'un seul tuple à la fois.

DELETE dans une vue

Vue mono-table

On peut faire des DELETE dans une vue mono-table : ils passeront dans la table correspondante, à condition que toutes les contraintes d'intégrité soient correctement prises en compte.

Vue multi-table

On peut pas faire de DELETE dans une vue multi-table.

Opération qui rendent une vue non modifiable

- Group by,
- Les Fonction de groupe : count(), sum(), etc.
- Distinct,
- Union (car il intègre un distinct)
- Les attributs calculés
- Jointure externe
- L'utilisation d'un « rownum » : ORACLE

Exemples

```
UPDATE emp3000
SET deptno=10
WHERE SAL > 3500;      // modification effectuée
```

```
UPDATE emp3000
SET sal=4000           // ça ne passera pas du fait du check
option
WHERE deptno > 10;
```

```
DELETE FROM emp3000
WHERE sal < 3000;      // ça ne passera pas du fait du check
option
```

8. Produits cartésiens et jointures en SQL 2

Les produits cartésiens

Syntaxe classique :

```
Select *  
from emp, dept ;
```

Le from peut être remplacé par :

➤ **JOIN**

```
Select *  
from emp join dept ;
```

ou

➤ **CROSS JOIN**

```
Select *  
from emp cross join dept ;
```

ou

➤ **INNER JOIN**

```
Select *  
from emp inner join dept ;
```

Les jointures naturelles

Syntaxe classique :

```
Select e.NE, e.nom, d.ville  
from emp e, dept d  
where e.nd = d.nd  
where ville = "PARIS";
```

Le from peut être remplacé par :

➤ **JOIN**

```
from emp e join dept d
```

ou

➤ **CROSS JOIN**

```
from emp e cross join dept d
```

ou

➤ **INNER JOIN**

```
from emp e inner join dept d
```

Le where peut être remplacé par :

➤ **ON**

```
Select e.NE, e.nom, d.ville  
from emp e join dept d  
on e.nd = d.nd  
where ville = "PARIS";
```

ou

➤ **USING**

```
Select e.NE, e.nom, d.ville  
from emp e join dept d  
using nd  
where ville = "PARIS";
```

Le from et le where peuvent être remplacés par :

➤ **NATURAL JOIN**

```
Select NE, nom, ville  
from emp e natural join dept d  
where ville = "PARIS";
```

Le natural join fait des restrictions de jointure avec tous les attributs de même nom des deux tables proposées. Donc avec le ND de emp et le ND de dept, ce qui conduit bien à une jointure naturelle.

Le problème du « natural join »

Présentation du problème

Quand on fait un « natural join », on en général une jointure entre la clé étrangère d'une première table et la clé primaire d'une deuxième table, à condition qu'elles aient le même nom. Mais si deux autres attributs ont le même nom, il y aura aussi une jointure sur ces deux attributs.

Par exemple, si on a un attribut « nom » dans la table des employés et un attribut « nom » dans la table des départements, la « natural join » conduira à une restriction du type :

```
Where emp.nd = dept.nd  
And emp.nom = dept.nom    ==>>  BUG !
```

Cause du problème

Le problème vient du fait que la jointure est faite à partir d'un nom d'attribut identique entre deux tables, ce qui est une considération conventionnelle et pas mathématique.

En réalité, le « natural join » est mal fait : il devrait faire la jointure automatiquement entre la une clé étrangère et la clé primaire référencée correspondante.

Du bon usage

➤ *Optimisation d'exécution*

Un « natural join » est équivalent à un ON ou un USING ou une jointure en SQL 1.

➤ *Optimisation de maintenance*

On évitera systématiquement le « natural join » qui peut conduire à des erreurs en maintenance : en effet, si on ajoute un attribut dans une table et qu'il existe déjà un attribut de même nom dans une autre table et qu'il existe déjà des « natural join » entre les deux tables, alors on aura des erreurs sur ces « natural join ».

Exemples avec plusieurs tables

On veut tous les livres actuellement empruntés d'un adhérent avec la date d'emprunt de ces livres

Syntaxe classique :

```
Select e.nl, l.editeur, a.na, a.nom, e.datemp
from emprunter e, adherents a, livres l
where e.na = a.na
and e.nl = l.nl
and e.dateretour is null ;
```

Avec des join

```
Select datemp, nl, editeur, na, nom
from emprunter e
join adherents a on (e.na=a.na)
join livres using(nl)
where e.dateretour is null;
```

ou

```
Select datemp, nl, editeur, na, nom
from emprunter e
join adherents a on (e.na=a.na and e.dateretour is null)
join livres using(nl);
```

On peut mettre indifféremment les restrictions spécifiques dans le on ou dans le where. De même pour les restrictions de jointure. L'intérêt de cette possibilité apparaîtra avec les jointures externes.

9. Jointures externes

Toutes les possibilités syntaxiques qu'on vient d'aborder n'apportent rien de plus par rapport à la syntaxe standard du SQL 1.

L'intérêt de cette syntaxe est d'ouvrir à la notion de jointure externe : outer join (par opposition à inner join).

Présentation de la jointure externe

Définition

Une jointure externe est une jointure qui permet de rajouter au résultat de la jointure les tuples éliminés d'une des deux tables de la jointure.

Jointure à gauche et jointure à droite

Selon que l'on rajoute les tuples de la première table ou de la deuxième (première ou deuxième dans l'ordre de déclaration), on parlera de jointure externe à gauche (première table, à gauche du join) ou à droite (deuxième table, à droite du join).

Syntaxe de la jointure externe à gauche

Après la jointure, on rajoute les tuples de la première table (celle de gauche) :

```
Select *  
  from t1 left outer join t2  
  on t1.att = t2.att;
```

ou encore, le « outer » étant facultatif :

```
Select *  
  from t1 left join t2  
  on t1.att = t2.att;
```

Syntaxe de la jointure externe à droite

Après la jointure, on rajoute les tuples de la deuxième table (celle de droite) :

```
Select *  
  from t1 right outer join t2  
  on t1.att = t2.att;
```

ou encore, le « outer » étant facultatif :

```
Select *  
  from t1 right join t2  
  on t1.att = t2.att;
```

Non commutativité de la jointure externe

A la différence de la jointure interne et du produit cartésien, la jointure externe n'est pas commutative :

```
Select *  
  from t1 right join t2  
  on t1.att = t2.att;
```

n'est pas équivalent à :

```
Select *  
from t1 right join t2  
on t1.att = t2.att;
```

Semi-commutativité de la jointure externe

Le « right join » est équivalent au « left join » si on inverse l'ordre des tables.

```
Select *  
from t1 right join t2  
on t1.att = t2.att;
```

est équivalent à :

```
Select *  
from t2 left join t1  
on t1.att = t2.att;
```

Remarque syntaxique

```
Select *  
from t1 outer join t2 // BUG !!! il faut préciser gauche ou  
droite  
on t1.att = t2.att;
```

La requête précédente ne veut rien dire! Il faut nécessairement préciser right ou left pour que la requête soit correcte.

Ordre des opérations élémentaires dans une jointure externe

```
Select *  
from t1 right join t2  
on t1.att = t2.att  
where tx.attx = valeur ;
```

L'ordre de traitements des opérations élémentaires est le suivant :

- 1) produit cartésien : **from**
- 2) restriction de jointure : **on**
- 3) jointure externe (ajout des tuples éliminés) : **right** (ou left)
- 4) restriction spécifique : **where**

On comprend ici qu'on pourra mettre des restrictions spécifiques dans le ON pour permettre

Les 3 principaux usages de la jointure externe

La jointure externe a trois usages principaux :

- Le premier usage concerne des **jointures naturelles** avec une table maître et une table jointe. Il consiste à rajouter les tuples éliminés de la table maître et à **s'intéresser à tous les tuples de la table maître**.
- Le deuxième usage concerne des **jointures naturelles** avec une table maître et une table jointe. Il consiste à rajouter les tuples éliminés de la table jointe et à **s'intéresser uniquement aux tuples éliminés de la table jointe. C'est une soustraction ensembliste**.
- Le troisième concerne les **jointures artificielles**. Il reprend le second dans le sens où il s'agit d'une soustraction ensembliste, mais à partir d'une jointure artificielle et avec jointure externe à gauche.

1^{er} usage (important) : s'intéresser à tous les tuples de la table maître : left join

Formalisme : jointure externe à gauche

La forme générale d'une jointure externe à gauche est :

```
Select TJ.NTJ, TJ.CS, TM.NTM, TM.CS
from table_maître TM left join table_jointe TJ
on TM.NTJ = TJ.NTJ;
```

Avec CS : clé significative.

En mettant la table maître en premier, on a une jointure à gauche.

Exemple

Question traitée : Afficher les employés avec leurs supérieurs hiérarchiques et les employés sans supérieurs hiérarchique :

```
Select e.ne, e.nom, echef.ne, echef.nom
from emp e left join emp echef
on e.nechef = echef.ne ;
```

Résultats :

```
+-----+-----+-----+-----+
| ne    | nom    | ne    | nom    |
+-----+-----+-----+-----+
| 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  |
| 7839  | KING   | NULL  | NULL   |
| 7844  | TURNER | 7698  | BLAKE  |
| 7876  | ADAMS  | 7788  | SCOTT  |
| 7900  | JAMES  | 7698  | BLAKE  |
| 7902  | FORD   | 7566  | JONES  |
| 7934  | MILLER | 7782  | CLARK  |
+-----+-----+-----+-----+
14 rows in set (0.00 sec)
```

Requête équivalente : UNION

La jointure externe à gauche classique permet de remplacer une union.

```
Select e.ne, e.nom, echef.ne, echef.nom
from emp e join emp echef
on e.nechef = echef.ne
union
Select ne, nom, null, null // on ajoute deux attributs à à emp
from emp
where nechef is null
;
```


Clé primaire

Clé primaire = Clé Primaire (Table Maître)

On retombe forcément sur la clé primaire de la table maître.

Nombre de tuples

Nb Tuples = Nb Tuples (Table Maître)

On retombe forcément sur le nombre de tuples de la table maître.

Du bon usage

Optimisation d'exécution

La jointure est théoriquement préférable aux opérateurs ensemblistes classiques, donc à l'union.

Optimisation de maintenance

La jointure externe « à gauche » est préférable à l'union qui est compliquée à mettre en œuvre.

A noter que l'usage de la jointure externe « à gauche » peut être généralisé par défaut pour toutes les jointures naturelles afin de ne pas éliminer les tuples dont la clé étrangère vaut NULL.

2^{ème} usage (important) : s'intéresser aux tuples éliminés de la table jointe : right join

Formalisme : jointure externe à droite

La forme générale d'une jointure externe à droite est :

```
Select TJ.*  
from table_maitresse TM right join table_jointe TJ  
on TM.NTJ = TJ.NTJ  
where TM.NTM is NULL;
```

En mettant la table maître en premier, on a une jointure à droite.

Exemple

Question traitée : quels sont les départements vides :

```
Select d.nd, d.nom, d.ville  
from emp e right join dept d  
on e.nd = d.nd  
where e.ne is NULL;
```

Résultats :

```
+-----+-----+-----+  
| nd | nomd          | ville |  
+-----+-----+-----+  
| 40 | OPERATIONS   | BOSTON |  
+-----+-----+-----+  
1 row in set (0.01 sec)
```

Requête équivalente : MINUS et INNERSECT, NOT IN, NOT EXISTS

La jointure externe à droite classique permet de remplacer les opérations ensemblistes MINUS et INTERSECT, et leurs équivalents en NOT IN et NOT EXISTS

➤ *Version MINUS (ne marche pas avec MySQL)*

```
Select *  
from dept  
minus  
select dept.*  
from emp e, dept d  
where e.ND = d.ND  
;
```

➤ *Version imbriquée NOT IN*

```
Select *  
from dept  
where ND not in (  
    select ND from emp  
);
```

➤ **Version imbriquée NOT EXISTS**

```
Select *  
  from dept  
 where not exists (  
     select * from emp  
     where emp.ND = dept.ND  
 );
```

Clé primaire

Pas de clé primaire.

Nombre de tuples

$Nb\ Tuples(Table\ jointe) \leq Nb\ Tuples \leq Nb\ Tuples(Table\ Maître) + Nb\ Tuples(Table\ jointe) - 1$

Si aucun tuple de la table maître ne pointe sur un tuple de la table jointe, la jointure naturelle donne un ensemble vide et la jointure externe rajoute tous les tuples de la table jointe.

Si tous les tuples de la table maître pointent sur le même tuple de la table jointe, la jointure naturelle donne tous les tuples de la table maître sur-informés. La jointure externe ajoute tous les tuples de la table jointe moins 1.

Du bon usage

Optimisation d'exécution

La jointure est théoriquement préférable à la requête imbriquée.

Optimisation de maintenance

La jointure externe étant un peu compliquée à comprendre, l'optimisation de la maintenance peut conduire à préférer le select imbriqué à la jointure externe.

3^{ème} usage (+ rare) : soustraction ensembliste à partir d'une jointure artificielle : left join

Exemple

Question traitée : tous les employés ayant un job différent de ceux du département 20

```
Select e1.*  
from emp e1 left join emp e2  
on e1.job = e2.job and e2.nd=20  
where e2.ne is NULL;
```

Principe :

➤ *D'abord, on inverse la question :*

tous les employés moins ceux qui ont le même job que ceux du département 20. C'est une jointure artificielle.

On croise tous les employés e1 avec ceux du département 20 (e2) et on garde ceux qui ont un job identique à un employé du département 20

La restriction de jointure et la restriction spécifique sont faites dans le « on ».

➤ *Ensuite, on prend le complément du résultat précédent :*

ce qui répond à la question. Le « left » rajoute les employés de e1 qui ont été éliminés : ceux qui ont un job différent de ceux du département 20.

➤ *Enfin, on restreint à ce seul complément avec le where.*

Il faut donc faire bien attention à mettre toutes les restrictions souhaitées dans le « on » avant que le « left » s'applique et rajoute les tuples éliminés.

Requête équivalente : MINUS, NOT IN, NOT EXISTS

La jointure externe à droite classique permet de remplacer les opérations ensemblistes MINUS et INTERSECT, et leurs équivalents en NOT IN et NOT EXISTS

➤ *Version imbriquée NOT IN*

```
Select *  
from emp  
where job not in (  
    select job from emp  
    where nd=20  
);
```

Du bon usage

Optimisation d'exécution

La jointure est théoriquement préférable à la requête imbriquée.

Optimisation de maintenance

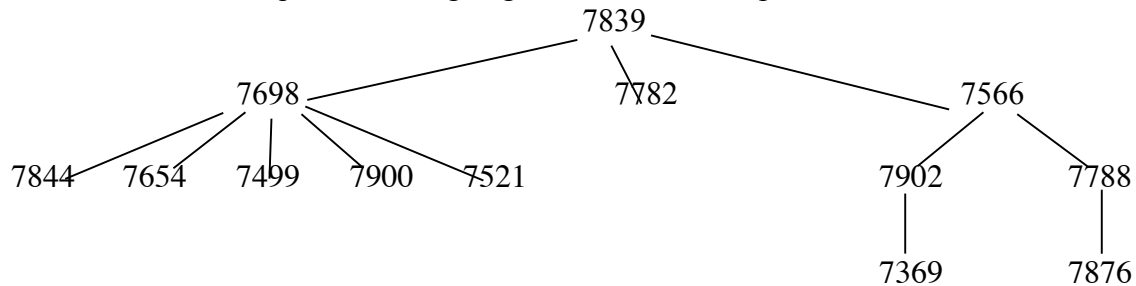
La jointure externe étant un peu compliquée à comprendre, l'optimisation de la maintenance peut conduire à préférer le select imbriqué à la jointure externe.

10. CONNECT BY PRIOR : gestion des arborescences

Présentation

Principes

Les tables avec des auto-jointures peuvent être considérées comme des structures d'arbres. On part de l'arbre suivant qui décrit l'organigramme d'une entreprise.



On souhaite pouvoir faire plusieurs choses :

1. Afficher tous les arbres : chaque nœud est considéré comme une racine.
2. Sélectionner un arbre
3. Afficher toutes les branches : de chaque nœud, on remonte à la racine.
4. Sélectionner une branche
5. Elaguer des branches
6. Elaguer des arbres

Solution SQL Server

Le type « HierarchyId » de SQL Server permet de gérer une structure d'arbre.

Solution ORACLE

ORACLE offre une clause : CONNECT BY PRIOR, qui va permettre de réaliser facilement les opérations précédentes.

Principes

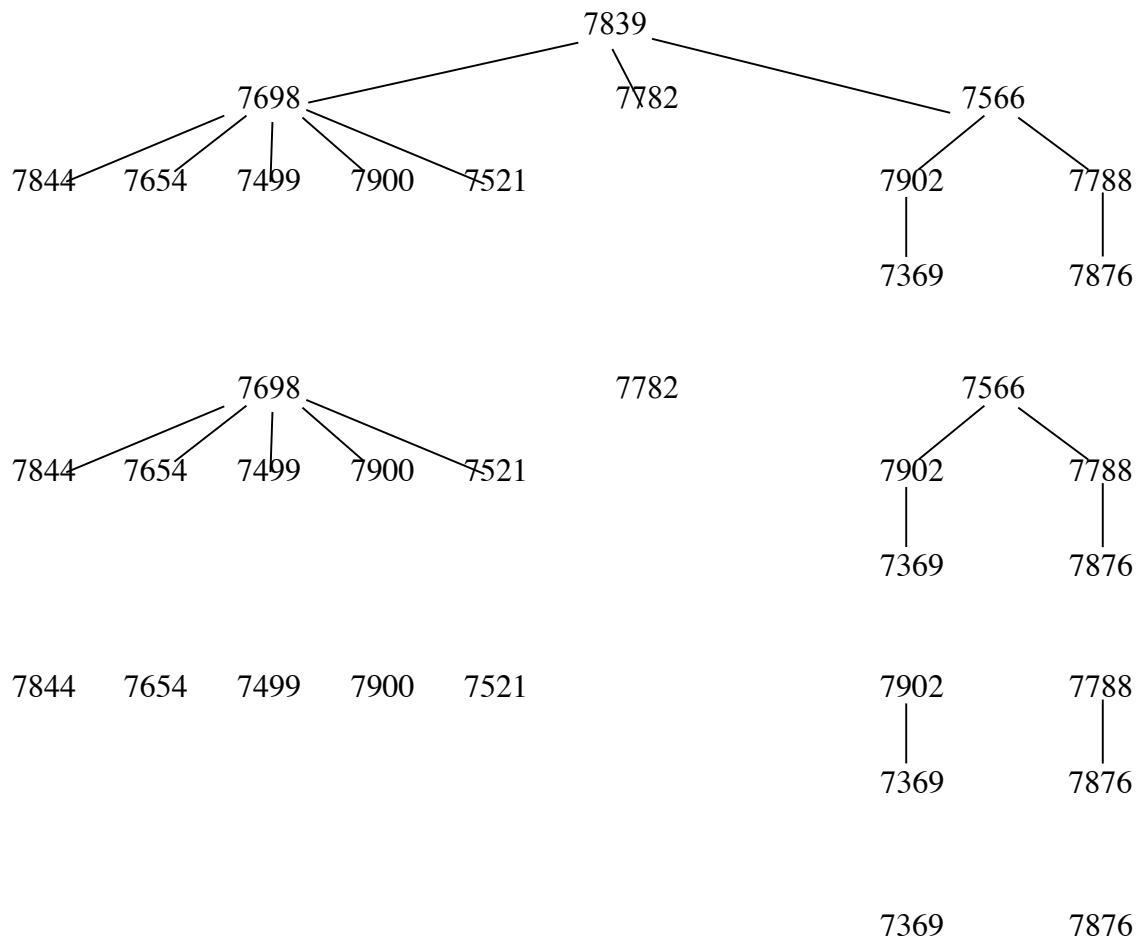
Chaque nœud est considéré comme une racine.

On a donc autant d'arbres que de nœuds.

Les arbres les plus simples sont constitués d'un seul nœud : une feuille de l'arbre d'origine.

Exemple

Dans notre exemple, on se retrouve avec 13 arbres :



Première approche

principes

```
select level, empno, mgr // level : niveau de l'enfant ds
l'arbre
from emp
connect by prior empno=mgr; // CONNECT BY PRIOR enfant =
parent
```

Principe du CONNECT BY PRIOR empno=mgr :

CONNECT BY PRIOR enfant = parent

« level » est une pseudo-colonne qui donne la « hauteur » de l'enfant dans l'arbre.

Le nœud de départ étant à 1, les feuilles de l'arbre d'origine ont toujours la hauteur la plus élevée.

résultats

LEVEL	EMPNO	MGR
-----	-----	-----
1	7902	7566
2	7369	7902
1	7788	7566
2	7876	7788
1	7654	7698
1	7499	7698
1	7900	7698
1	7521	7698
1	7844	7698
1	7876	7788
1	7698	7839
2	7654	7698
2	7499	7698
2	7900	7698
2	7521	7698
2	7844	7698
1	7782	7839
1	7566	7839
2	7902	7566
3	7369	7902
2	7788	7566
3	7876	7788
1	7369	7902
1	7839	
2	7698	7839
3	7654	7698
3	7499	7698
3	7900	7698
3	7521	7698
3	7844	7698
2	7782	7839
2	7566	7839
3	7902	7566
4	7369	7902
3	7788	7566
4	7876	7788

36 ligne(s) sélectionnée(s).

Présentation en arbres

principes

```
select lpad(' ',4*level-4) || empno arbre
from emp
connect by prior empno=mgr;
```

La fonction LPAD (chaîne, nb) fabrique une chaîne de « nb » caractères dont ceux de droite seront constitué par « chaîne ». Ici, elle nous permet de décaler l’affichage de « empno » en fonction du « level ».

Ainsi, on peut voir les 13 racines des 13 arbres possibles.

On voit aussi les 8 feuilles (mais qui apparaissent plusieurs fois en tant que feuilles)

résultats

```
ARBRE
-----
7902
  7369
7788
  7876
7654
7499
7900
7521
7844
7876
7698
  7654
  7499
  7900
  7521
  7844
7782
7566
  7902
    7369
  7788
    7876
7369
7839
  7698
    7654
    7499
    7900
    7521
    7844
  7782
  7566
    7902
      7369
    7788
      7876

36 ligne(s) sélectionnée(s).
```


ELEMENTS THEORIQUES DE L'ALGEBRE

RELATIONNELLE

1. Modèle relationnel

Domaine

Un domaine c'est un ensemble de valeurs (que peut prendre un prédicat ou attribut). Il est défini en intension ou en extension. Il est non vide.

Exemples de domaine

- l'alphabet, domaine défini en extension;
- une chaîne de 20 caractères, défini en intension (toutes les combinaisons possibles de lettres entre 1 et 20 caractères).
- les entiers, domaine défini en intension;
- les entiers compris entre 0 et 100, domaine défini en extension;
- l'ensemble de couleurs {bleu, rouge, vert, marron}, domaine défini en extension.

Relation (table)

Une relation n-aire (ou table) est un sous-ensemble du produit cartésien d'une liste de domaines D_1, D_2, \dots, D_n .

$$R = \text{sous ensemble de } D_1 \times D_2 \times \dots \times D_n$$

Une relation peut être vue comme un tableau à deux dimensions dont les colonnes correspondent aux domaines (et donc aux attributs) et les lignes correspondent aux éléments individuels.

Attribut

Un attribut est une fonction entre une relation et un domaine auquel on ajoute la valeur NULL, cette valeur voulant dire « non défini ».

$$A_i : R \rightarrow D_i \cup \{\text{NULL}\}$$

Une relation R définie sur n attributs est notée :

$$R(A_1, A_2, \dots, A_n)$$

Un attribut c'est une colonne d'une relation caractérisée par un nom. L'ordre des colonnes dans la relation n'a pas d'importance.

Tuple

Un tuple est un n-uplet du produit cartésien d'une liste de domaines.

Le tuple, c'est une ligne de la table. L'ordre des lignes dans la table n'a pas d'importance.

Exemple :

Un tuple de la table des employés, c'est un sext-uplet du produit cartésien entre le domaine des n°, des noms, des fonctions, des dates, des salaires et des commissions.

Clé

Définition

La clé d'une table est un **sous-ensemble d'attributs qui détermine tous les autres attributs**. Ce sous-ensemble est tel que tous les attributs sont nécessaires pour former la clé.

Soit R une relation n -aire : $R(A_1, A_2, \dots, A_n)$

Soit l'ensemble d'attributs $\{A_i \dots A_k\}$ avec $\{i, \dots, k\} \in [1 : n]$

Soit f la fonction entre R et $D_i \times \dots \times D_k$. $f : R \rightarrow D_i \times \dots \times D_k$

L'ensemble d'attributs $\{A_i \dots A_k\}$ est une clé de R si et seulement si :

$$\forall x, y \in R, x \neq y \Rightarrow f(x) \neq f(y)$$

Obligation de la clé

Toute relation possède au moins une clé.

Donc, il n'y a jamais de doublons dans une relation.

Clé irréductible

Déf. (clé irréductible) : une clé C sur R est dite irréductible si et seulement si pour tout autre ensemble d'attributs $D \subset C$, D n'est pas une clé.

Clé minimale

Déf. (clé minimale) : une clé C sur R est dite minimale s'il n'existe aucune autre clé sur R impliquant un moins grand nombre d'attributs.

Clé candidate

Déf. (clé candidate) : toute clé de taille minimale est une clé candidate.

Clé primaire

Déf. (clé primaire) : la clé unique choisie par le concepteur de la base parmi les clés candidates est clé primaire.

Clé étrangère

Déf. (clé étrangère) : une clé étrangère est un attribut qui fait référence à un attribut clé primaire.

Dépendance fonctionnelle

Un attribut Y (ou premier groupe d'attributs) dépend fonctionnellement d'un attribut X (ou second groupe d'attributs), si étant donné une valeur de X, il lui correspond une valeur unique de Y (et ceci quel que soit l'instant considéré).

On dit que X détermine fonctionnellement Y et on note :

$$X \rightarrow Y$$

Concrètement, les clés d'une relation déterminent fonctionnellement tous les autres attributs de la relation.

2. Algèbre relationnelle

Présentation

2 opérateurs élémentaires unaires

2 opérateurs spécifiques : projection et restriction

5 opérateurs élémentaires binaires

5 opérateurs ensemblistes : union, intersection, différence, division, produit cartésien

1 opérateur composé fondamental

1 opérateur composé : jointure (composé d'un produit cartésien et d'une restriction)

résultat

Le résultat de l'application d'un opérateur sur une ou plusieurs relations est une relation.

Les 2 opérateurs unaires : projection et restriction

Projection

Projeter des attributs d'une relation consiste à ne conserver que ces attributs dans la relation résultante et donc se séparer des autres.

➤ *Formulation mathématique*

Soit R une relation n -aire : $R(A_1, A_2, \dots, A_n)$

Soit l'ensemble d'attributs $L = \{A_i, \dots, A_k\}$ avec $\{i, \dots, k\} \in [1 : n]$

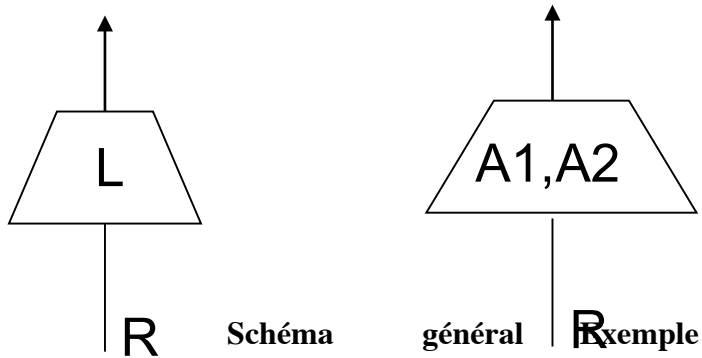
$$\Pi_L(R) = R'(A_i, \dots, A_k)$$

$$\text{ou encore : } \Pi_{A_i..A_k}(R) = R'(A_i, \dots, A_k)$$

On peut aussi écrire :

$$\text{Proj}(R ; A_i, \dots, A_k)$$

➤ **Représentation graphique**



A1	A2
a	b
a	a



A1	A2	A3
a	b	c
a	a	b

de

Restriction

La restriction consiste à ne conserver que certains tuples de la relation départ.

➤ **Formulation mathématique**

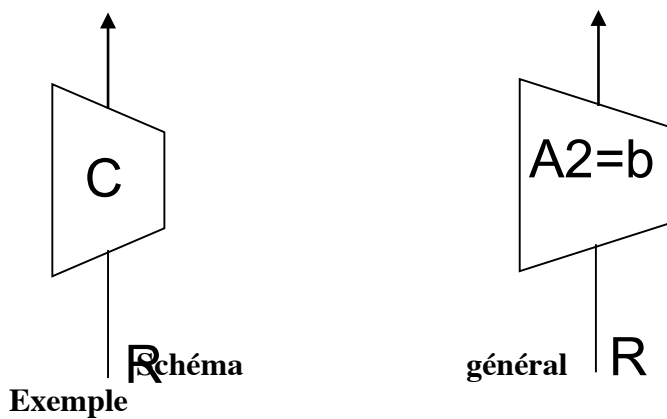
Soit R est une relation et C une condition, alors

$$\sigma_C(R) = \{ x \in R \text{ et } C(x) \}$$

On peut aussi écrire :

$$\text{Rest}(R ; C)$$

➤ **Représentation graphique**



A1	A2	A3
a	b	c



A1	A2	A3
a	b	c
a	a	b

Union, intersection, différence

➤ *Formulation mathématique*

Union, intersection et différence sont des opérateurs binaires.

L'union et l'intersection sont des opérateurs commutatifs, la différence un opérateur non commutatif.

Les relations doivent avoir les mêmes attributs.

Soit R1 et R2 deux relations, alors

Union : $R1 \cup R2 = \{ x \text{ tel que } x \in R1 \text{ ou } x \in R2 \}$

Intersection : $R1 \cap R2 = \{ x \text{ tel que } x \in R1 \text{ et } x \in R2 \}$

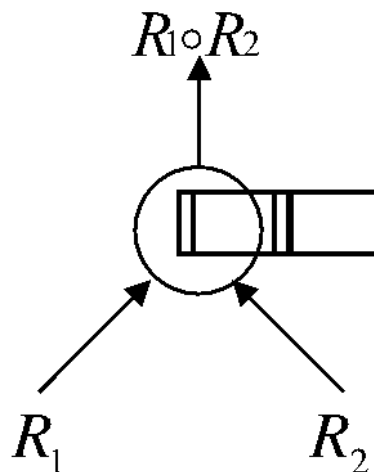
Différence : $R1 - R2 = \{ x \text{ tel que } x \in R1 \text{ et } x \notin R2 \}$

On peut aussi écrire :

Union(R1, R2)
Etc.

➤ *Représentation graphique*

Avec « o » pour un opérateur parmi l'union, l'intersection ou la différence.



Division

➤ *Formulation mathématique*

La division est un opérateur binaire non commutatif.

Soit B une relation n-aire : B(A1, A2, ..., An)

Soit A une relation m-aire : A(A1, A2, ..., An, An+1, ..., Am) avec $m > n$.

A / B est une relation R définie par les attributs (An+1, ..., Am)

$$A / B = \{ x \in R \text{ tel que } \forall y \in B, x \times y \in A \}$$

On peut aussi écrire :

$$\text{Division}(A, B)$$

➤ *Représentation graphique*

Même principe que pour l'union, l'intersection et la différence.

➤ *Exemple*

A	X	/	X	=	A
A	Y		Y		
A	Z				
B	X				
C	Y				

Produit cartésien

➤ *Formulation mathématique*

Le produit cartésien est un opérateur binaire commutatif.

Soit R1 et R2 deux relations, alors

$$R1 \times R2 = \{ (x, y) \text{ tel que } x \in R1 \text{ et } y \in R2 \}$$

Remarque : le produit cartésien est le seul opérateur qui permet de produire une table regroupant les attributs de deux tables.

On peut aussi écrire :

$$PC (R1, R2)$$

➤ *Représentation graphique*

Même principe que pour l'union, l'intersection et la différence.

➤ *Exemple*

A	A	\times	X	=	A	A	X
A	B		Y		A	A	Y
			Z		A	A	Z
					A	B	X
					A	B	Y
					A	B	Z

Formulation mathématique

Une jointure est un produit cartésien associé à une restriction de jointure.

Soit A une relation : A(A1, A2, ..., An)

Soit B une relation : B(B1, B2, ..., Bm)

Une condition de jointure entre A et B est une condition qui met en jeu au moins un Ai et un Bi.

Soit C une condition de jointure entre A et B, alors

$$R1 \bowtie_C R2 = \sigma_C (R1 \times R2) = \{ (x, y) \text{ tel que } x \in R1 \text{ et } y \in R2 \text{ et } C(x,y) \}$$

On peut aussi écrire :

$$J (R1, R2; C)$$

Distinction entre jointure naturelle et jointure artificielle

Attention : cette distinction ne correspond pas au modèle relationnel standard.

Une jointure naturelle est une jointure dont la condition est de la forme :

$$R1.\text{cléEtrangère} = R2.\text{cléPrimaire}$$

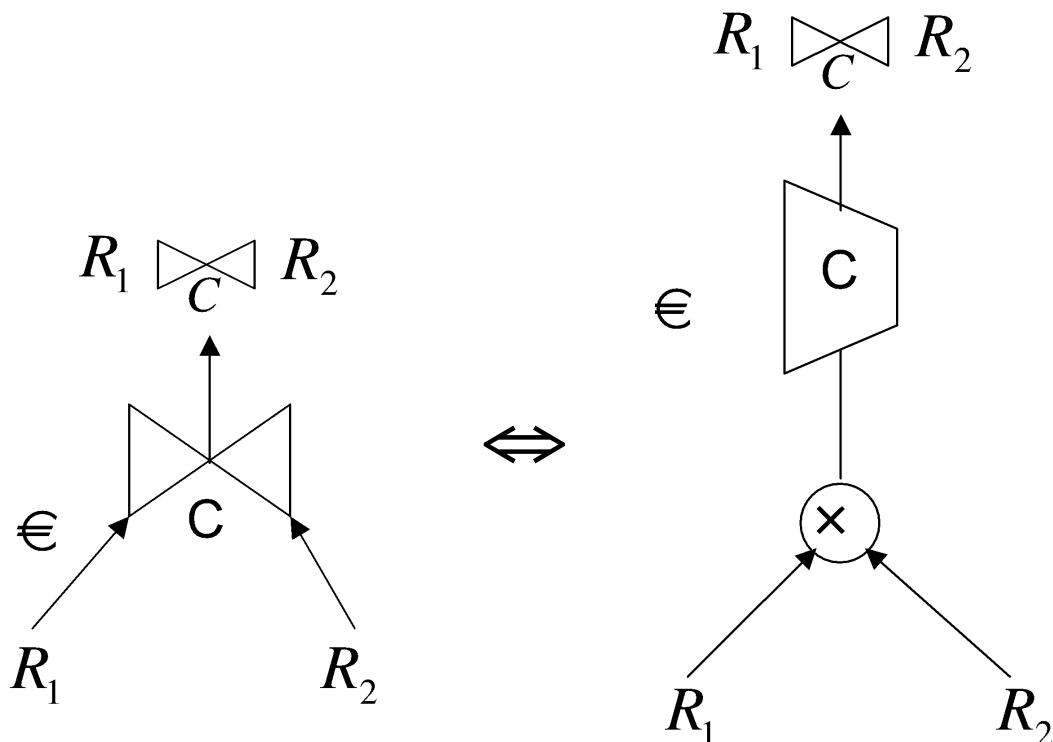
Elle produit une relation dont la clé primaire est celle de R1.

On peut écrire :

$$JN (R1, R2)$$

Une jointure artificielle est une jointure qui n'est pas naturelle.

Représentation graphique Une jointure artificielle est une jointure qui n'est pas naturelle.



SGBD – SQL - TP : SELECT MULTI-TABLES

Nom :

Encodage du fichier texte en UTF8 - BOM

Présentation et préparation du travail

- Le sujet de TP est dans un fichier texte téléchargeable : MonNom_TP_SQL_multitables.txt
- Changez le nom du fichier : remplacez MonNom par votre nom : exemple : Liaudet_TP_SQL_multitables.txt
- Dans le fichier, à la première ligne, mettez votre nom.
- Vous mettrez ensuite vos réponses directement dans le fichier.
- Il sera envoyé à l'adresse mail : liaudet.bertrand@wanadoo.fr

EXERCICE 1 : charger les tables de la base de données - EmployesTP02.sql

Présentation

On travaille sur la table suivante qui se trouve dans le fichier EmployesTP02.sql

EMP(NE, NOM, JOB, DATEMB, SAL, COMM, #ND, *NEchef)

- NE numéro de l'employé. **Clé primaire.**
- NOM nom de l'employé.
- FONCTION intitulé du poste occupé.
- DATEMB date d'embauche.
- SAL salaire de l'employé.
- COMM commission (part de salaire variable).
- #ND n° du département dans lequel travaille l'employé. **Clé étrangère.**
- *NEchef n° d'employé du chef de l'employé. **Clé étrangère réflexive.**

DEPT(ND, NOM, VILLE)

- ND numéro des départements de l'entreprise. **Clé primaire.**
- NOM nom des départements de l'entreprise.
- VILE nom des villes où sont situés les départements.

Les requêtes

1. Téléchargez le script de création de la BD : EmployesTP02.sql
2. Lancez ce script de création de la BD.
3. Vérifiez que la table est créée (visuellement dans WAMP, ou avec la commande « desc »)
4. Affichez tous les tuples des 2 tables (avec WAMP ou avec un select).
5. Regardez le fonctionnement du NULL dans l'insert des employés.
6. Regardez le fonctionnement du last_insert_id() dans l'insert des employés. Qui est le chef de ADAMS et le chef de SMITH ? Pourquoi ?

EXERCICE 2 : Jointures naturelles - Chapitres 1 et 2 du cours

Travail à faire

- Répondre à toutes les questions ci-dessous en SQL1 et en SQL2.
- Le fichier répond déjà aux 2 premières questions du TP : vous devez utiliser le même formalisme pour les autres questions.
- Pour chaque question vous devez mettre :
 1. L'intitulé de la question
 2. Le select de la réponse
 3. La CP de la table résultat de chaque question

Priorité des questions

- Dans un premier temps, répondez uniquement aux questions ne commencent pas par (*).
- Ensuite traitez les questions avec (*).

Attention !

- Vous ne devez projeter que les attributs nécessaires : CP, CS, AD, AR – Tri devant ou derrière (Clé Primaire, Clé Significative, Attributs Demandés, Attributs de restriction).
- Vous devez préciser la clé primaire de chaque table de résultats.

Outils

- Utilisez un éditeur efficace : notepad++, sublime text : qui fasse de la coloration syntaxique. Mettez une extension .sql à vos fichiers pour avoir la coloration syntaxique.
- Vous pouvez travailler dans la calculatrice mysql ou celle de phpMyAdmin.

Les requêtes : 2 versions à chaque fois : SQL1 et SQL2

Questions de base

1. Tous les employés avec tous leurs attributs et toutes les informations de leurs départements.
2. Tous les employés de Dallas.
3. Combien d'employés à Dallas ?
4. Salaire moyen à Dallas.

Questions techniques

5. Faire le produit cartésien de la table Employés et la table Départements en projetant tous les attributs.
 - a. Combien y a-t-il d'attributs dans la table résultat ?
 - b. Combien y a-t-il de tuples dans la table résultat ?
 - c. Quelles formules permettent de trouver les résultats ?
6. Faire la jointure naturelle de la table Employés et la table Départements en projetant tous les attributs.
 - a. Combien y a-t-il d'attributs dans la table résultat ?
 - b. Combien y a-t-il de tuples dans la table résultat ?
 - c. Quelles formules permettent de trouver les résultats ?

Suite des questions

7. Tous les employés du département RESEARCH. Faire le graphe de la question.
8. Tous les analystes du département RESEARCH. Faire le graphe de la question.
9. Tous les employés qui gagnent plus que 1500 et qui travaillent à NEW YORK. Faire le graphe de la question.
10. Nombre d'employés et moyenne des salaires des employés travaillant à DALLAS
11. Moyenne des commissions des vendeurs du département SALES

Questions sur les chefs de base

12. Tous les employés avec leurs chefs. Faire le graphe de la question.
13. Combien y a-t-il de tuples dans la question précédente ? Pourquoi ?
14. Tous les employés dont le chef est à DALLAS. Faire le graphe de la question.

(*) Questions techniques sur les chefs de base

15. Faire le produit cartésien de la table Employés avec elle-même.
 - a Combien y a-t-il d'attributs dans la table résultat ?
 - b Combien y a-t-il de tuples dans la table résultat ?
 - c Quelles formules permettent de trouver les résultats ?
16. Faire la jointure naturelle de la table Employés avec elle-même.
 - a Combien y a-t-il d'attributs dans la table résultat ?
 - b Combien y a-t-il de tuples dans la table résultat ?
 - c Quelles formules permettent de trouver les résultats ?

Suite des questions sur les chefs

17. Tous les employés ne travaillant pas dans le même département que leur chef. Faire le graphe de la question.
18. Tous les employés embauchés avant leur chef. Faire le graphe de la question.
19. Donner la liste des numéros de chef, le nombre de leur subordonnés et le salaire moyen des subordonnés (préparation à la question suivante, pas de jointure dans cette question).
20. Donner la liste de tous les supérieures hiérarchiques avec leur salaire, le nombre de leur subordonnés et le salaire moyen, min et max de leurs subordonnés. On entend par salaire le salaire et les commissions. On tri les résultats par nombre de subordonnés et moyenne des salaires décroissants.
21. (*) Tous les chefs avec la liste de leurs subordonnés.

Plus de 2 tables

22. Tous les employés avec le nom de leur département et le nom de leur chef ainsi que le nom du département de leur chef.
 - a Combien y a-t-il de tuples dans la table résultat ? Pourquoi ?
23. Tous les employés ne travaillant pas dans la même ville que leur chef.
24. Les Clerks dont le chef n'est pas de New York. On veut aussi le nom du département des Clerks.

EXERCICE 3 : Jointures artificielles et requêtes imbriquées - Chapitres 3 à 7 du cours

Travail à faire

La même chose que pour l'exercice 2, à la suite dans le même fichier.
Quand on demande 2 versions, il faut proposer une jointure et une imbriquée.

On traite uniquement les questions en gras (les 7 premières)

Les suivantes, avec un (*) au début, sont facultatives.

Les requêtes

1. **Tous les employés travaillant dans un département qui contient au moins un 'ANALYST'.**
Ecrire 2 versions.
2. **Tous les employés ayant le même job que les employés du département 30.**
Ecrire 2 versions.
3. **Tous les noms et dates d'embauche des employés embauchés avant BLAKE.**
Ecrire 2 versions.
4. **Tous les employés ayant le même chef que ALLEN**
Ecrire 2 versions.
5. **Tous les employés n'ayant pas de subordonnées.**
Ecrire 2 versions : un minus théorique (n'existe pas dans MySQL) et une imbriqué.
6. **Tous les employés qui gagnent plus que la moyenne, classés par salaire croissant.**
Ecrire 2 versions : une version imbriquée dans le from, une version imbriquée dans le where.
7. **Tous les départements qui contiennent plus d'employés que le nombre moyen d'employés par ville.**
Faire deux versions imbriqués (avec et sans imbrications dans le where)
Faire plusieurs versions avec des vues.

8. (*) **Tous les départements vides avec leurs noms et villes.**
Ecrire 2 versions : un minus théorique (n'existe pas dans MySQL) et une imbriqué.
9. (*) **Tous les employés n'ayant pas le même chef que ALLEN**
Ecrire 2 versions (jointure artificielle et NOT IN)
10. (*) **Tous les employés ayant le même job et le même chef que MARTIN.**
Ecrire 3 versions.
11. (*) **Tous les employés travaillant à Chicago et ayant le même job qu'ALLEN.**
Ecrire 3 versions.
12. (*) **Tous les employés du département RESEARCH embauchés la même année que quelqu'un du département SALES**
Ecrire 3 versions.
13. (*) **Tous les employés embauchés avant tous les employés du département 10.**
Ecrire 2 versions.
Pour vérifier on affichera tous les employés du 10 ordonnés par date d'embauche.

EXERCICE 4 : Jointures externes et SQL2 - Chapitres 8 et 9 du cours

Travail à faire

La même chose que pour l'exercice 3, à la suite dans le même fichier.
Quand on demande 2 versions, il faut proposer une jointure et une imbriquée.

On traite uniquement les questions en gras (les 7 premières)
Les suivantes, avec un (*) au début, sont facultatives.

Les requêtes

Questions techniques

1. Faites une jointure naturelle entre « emp » et « dept » avec un « **using** » pour avoir chaque employé avec son département
2. Faites une jointure naturelle entre « emp » et « dept » avec un « **on** ».
3. Ajoutez un **inner** à la question 2. Que constatez-vous ?
4. Ajouter un **outer** à la question 2. Que constatez-vous ?
5. Faites une jointure naturelle entre « emp » et « dept » avec un « **natural join** » : que constatez-vous ?
6. En partant de la question 2, faites une **jointure externe à gauche (left)** entre « emp » et « dept ». Que constatez-vous ?
7. En partant de la question 2, faites une **jointure externe à droite(right)** entre « emp » et « dept ». Que constatez-vous ?

Questions fonctionnelles

8. Quels sont les départements vides ?
9. Quels sont les employés qui ne sont pas supérieurs hiérarchiques ?
10. Listez tous les employés avec leur salaire et celui de leur chef. On l'identifiant et le nom des chefs. On listera aussi les employés qui n'ont pas de chefs.
11. Tous les employés n'ayant pas le même chef que ALLEN

Questions techniques

12. Faites une jointure naturelle entre « emp » et « emp » avec un « **using** » pour avoir les chefs.
13. Faites une jointure naturelle entre « emp » et « emp » avec un « **on** ».
14. Ajoutez un **inner** à la question 13. Que constatez-vous ?
15. Ajouter un **outer** à la question 13. Que constatez-vous ?

16. Faites une jointure naturelle entre « emp » et « emp » avec un « **natural join** » : que constatez-vous ?
17. En partant de la question 13, faites une **jointure externe à gauche (left)** entre « emp » et « emp ». Que constatez-vous ?
18. En partant de la question 13, faites une **jointure externe à droite (right)** entre « emp » et « emp ». Que constatez-vous ?