SQL (mysql)

Université de Caen-Normandie

Bruno Crémilleux

SQL : Structured Query Language

Créé en 1974, normalisé depuis 1986.

- **LDD** : Langage de Définition des Données : `CREATE`, `ALTER`
  Créer le schéma d'une base de données.
- **LMD** : Langage de Manipulation des Données :
  `SELECT`, `INSERT`, `UPDATE`, ...
  Interroger, insérer, modifier, supprimer des données.
- **LCD** : Langage de Contrôle des Données : `GRANT`,...
  Autorisation, sécurité, accès concurrents.

Langage "déclaratif" : l'utilisateur indique les données qui l'intéressent via une assertion (description formelle de l'information recherchée, sans spécification de chemin).

Calcul relationnel sur n-uplets

Univers de SQL : c'est l'ensemble des n-uplets des relations de la BD.

Variable typée : une variable typée par une table prend ses valeurs dans les lignes de cette table.

Exemple avec la BD "Commandes" (cf. TD et TP) :

`CLIENT C` crée la "variable client" `C` qui peut prendre pour valeur un des n-uplets de la table `CLIENT`.

Exemple avec le 1er n-uplet de `CLIENT` :

```
C.RefC = 1  C.NomC = 'Goffin' C.Ville = 'Namur'  C.Cat = 'B2'
```

Mêmes formules de sélection que pour l'algèbre relationnelle : connecteurs logiques ($\wedge$, $\vee$, $\neg$), opérateurs de comparaison, opérateurs arithmétiques.

mysql: premiers pas (1/2)

À partir d'un terminal :

```
mysql -h mysql.info.unicaen.fr -u LOGIN -p
```

exemple :

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
[...]
MariaDB [(none)]>
```

valeurs des paramètres de connexion : répertoire `~/Protected/mysql.txt` de votre home.

informations à https://faq.info.unicaen.fr/bdd

mysql: premiers pas (2/2)

La première fois, il faut créer sa base :

sous `mysql` :

```
MariaDB [(none)]> CREATE DATABASE LOGIN_bd ;
```

puis se connecter à sa base :

```
MariaDB [(none)]> use LOGIN_bd
```

Pour les connexions suivantes à la base `LOGIN_bd` :

à partir d'un terminal :

```
mysql -h mysql.info.unicaen.fr -u LOGIN -p LOGIN_bd
```

mysql(1/3)

Deux types de commandes :

- commandes de `mysql`:
  - `\?` ou `help` : aide
  - `\q` : quitter
  - `\. FILE` (ou `source FILE`) : exécute le script sql `FILE`
  - `\! COMMAND` : exécute la commande shell `COMMAND`
  - `\T FILE` : redirige la sortie dans le fichier `FILE`
  - ...
- commandes SQL : `SELECT`, `CREATE`, `INSERT`,...

  Une requête SQL se termine par un ;

Bonne habitude de travail : préparer les requêtes via un éditeur de texte et charger le script contenant les requêtes avec `\.` (ou `source`)

mysql(2/3)

Liste des tables : `MariaDB [LOGIN_bd]> show tables ;`

```
+-------------------------+
| Tables_in_cremilleux_bd |
+-------------------------+
| CLIENT                  |
| COMMANDE                |
| DETAIL                  |
| PRODUIT                 |
+-------------------------+
```

Schéma d'une table : `MariaDB [LOGIN_bd]> describe CLIENT ;`

```
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| RefC  | int(11)     | NO   | PRI | NULL    |       |
| NomC  | varchar(20) | NO   |     | NULL    |       |
| Ville | varchar(20) | NO   |     | NULL    |       |
| CAT   | varchar(2)  | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

mysql(3/3)

Contenu d'une table : `MariaDB [LOGIN_bd]> SELECT * from CLIENT ;`

```
+------+----------+----------+------+
| RefC | NomC     | Ville    | CAT  |
+------+----------+----------+------+
|    1 | GOFFIN   | Namur    | B2   |
|    2 | HANSENNE | Poitiers | C1   |
|    3 | MONTI    | Geneve   | B2   |
...
15 rows in set (0,00 sec)
```

Nous reviendrons sur le SELECT

Commentaire :

- une ligne : # ou --
- plusieurs lignes : /* ... */

Création d'une table

`CREATE TABLE...` :
description de chaque attribut :

- nom de l'attribut (une chaîne de caractères)
- type de l'attribut : entier, réel, chaîne, date,...
- propriétés de l'attribut : clé, `NOT NULL`, contraintes,...

```
CREATE TABLE DETAIL(
     RefCOM INT NOT NULL,
     RefP  VARCHAR(5) NOT NULL,
     Quantite INT NOT NULL,
     PRIMARY KEY (RefCOM, RefP)
) ;
```

`PRIMARY KEY (RefC)` : déclaration d'une clé (identifiant unique pour chaque n-uplet)

La table est vide après sa création.

Insertion de n-uplets

Deux possibilités :

- via un script SQL (commande `INSERT`) :

```
INSERT INTO CLIENT VALUES (1, 'GOFFIN', 'Namur', 'B2') ;
INSERT INTO CLIENT VALUES (2, 'HANSENNE', 'Poitiers', 'C1') ;
...
```

- via le chargement d'un fichier texte (cf. TP) :

  syntaxe `mysql` : (utiliser `\copy` en `postgres`)

```
LOAD DATA LOCAL INFILE "client.dat" INTO TABLE CLIENT ;
```

  où `client.dat` contient :

```
1 GOFFIN Namur B2
2 HANSENNE Poitiers C1
...
```

Il est possible que vous deviez explicitement activer lors de votre connexion la possibilité de chargement d'un fichier, cf. informations à https://faq.info.unicaen.fr/bdd

Création d'une table à partir d'une requête d'interrogation

```
CREATE TABLE NomTable AS
SELECT [...] ;
```

La table est remplie après sa création.

Utilisation : lorsque la base de données contient les informations sur la nouvelle table (e.g., reconstruction, mise à jour).

Possibilité d'insertion à partir d'une sélection :

```
INSERT INTO NomTable SELECT [...] ;
```

Exemple de requête
(en calcul des n-uplets et SQL)

Noms des clients qui habitent Namur :

Calcul des n-uplets : $\{$C.NomC de CLIENT C où Ville $=$ 'Namur'$\}$

En SQL :

```
SELECT C.NomC
FROM CLIENT C
WHERE C.Ville = 'Namur' ;
```

Autres façons d'écrire en SQL :

```
SELECT CLIENT.NomC
FROM CLIENT
WHERE CLIENT.Ville = 'Namur' ;
```

On confond le nom et le type de la variable (une seule variable `CLIENT`)

```
SELECT NomC
FROM CLIENT
WHERE Ville = 'Namur' ;
```

SQL est tolérant (il se débrouille avec le contexte si il n'y a pas d'ambiguïté)

Exemple de requête
(en calcul des n-uplets et SQL)

Les types de produits :

Calcul des n-uplets : $\{P.TypeP \text{ de } PRODUIT\ P\}$

En SQL :
```
SELECT DISTINCT P.TypeP
FROM PRODUIT P ;
```

`DISTINCT` : pour éliminer les doublons (par défaut, SQL est paresseux)

```
+----------+
| TypeP    |
+----------+
| Cheville |
| Cheville |
| Cheville |
| Clou     |
| Clou     |
| Planche  |
| Planche  |
+----------+
```
Sans `DISTINCT`

```
+----------+
| TypeP    |
+----------+
| Cheville |
| Clou     |
| Planche  |
+----------+
```
Avec `DISTINCT`

Exemple de requête
(en calcul des n-uplets et SQL)

Toutes les informations sur le client qui a effectué la commande de référence numéro 4 :

Calcul des n-uplets :

$\{$C.* de CLIENT C où $\exists$ COMMANDE COM (COM.RefC $=$ C.RefC $\wedge$ COM.RefCom $=$ 4)$\}$

En SQL :

```
SELECT C.*
FROM CLIENT C, COMMANDE COM
WHERE COM.RefC = C.RefC
AND COM.RefCom = 4 ;
```

C.* : tous les attributs de C.

RefC	NomC	Ville	CAT
9	PONCELET	Toulouse	B2

Exemple de requête
(en calcul des n-uplets et SQL)

Pour chaque nom de client, les références des produits qu'il a commandés :

Calcul des n-uplets :

$\{$C.NomC, D. RefP de CLIENT C, DETAIL D où $\exists$ COMMANDE COM
(C.RefC = COM.RefC $\wedge$ COM.RefCom = D.RefCom)$\}$

En SQL :

```
SELECT DISTINCT C.NomC, D.RefP
FROM CLIENT C, COMMANDE COM, DETAIL D
WHERE C.RefC = COM.RefC
AND COM.RefCom = D.RefCom ;
```

`DISTINCT` car un client peut avoir commandé le même produit dans 2 commandes différentes.

En algèbre relationnelle : `(C×COM×D):((C.RefC = COM.RefC) ∧ (COM.RefCom = D.RefCom))[NomC, RefP]`

Produit cartésien - sélection - projection

Forme générale d'une requête SQL

```
(3) SELECT [DISTINCT] A1, A2,..., Am (liste d'attributs
                                      et d'expressions calculées)
(1) FROM R1, R2,..., Rn (liste de relations)
(2) WHERE F  (expression de sélection)
```

Ordre d'exécution de la requête : (1) - (2) - (3)

En algèbre relationnelle :

$((R1 \times R2 \times \ldots \times Rn) : F)[A1, A2, \ldots, Am]$

Le SELECT correspond à une projection

Cas particulier : produit cartésien

```
SELECT *
FROM R1, R2,...,Rm
```

Les tables ne sont pas forcément distinctes.

Exemple : paires de noms de clients habitants la même ville.

```
C1 = Client ; C2 = Client ;

SELECT DISTINCT C1.NomC, C2.NomC
FROM CLIENT C1, CLIENT C2
WHERE C1.Ville = C2.Ville
AND C1.NomC < C2.NomC ;
```

En algèbre relationnelle :

$$(C1 \times C2):(C1.Ville = C2.Ville \text{ et } C1.NomC < C2.NomC)\ [C1.NomC, C2.NomC]$$

Tri de l'affichage : ORDER BY

Références, types et prix des produits commandés en 2006. Le résultat sera ordonné selon l'ordre croissant des types de produits puis l'ordre décroissant des prix :

```
SELECT DISTINCT P.TypeP, P.Prix, P.RefP
FROM PRODUIT P, DETAIL D, COMMANDE COM
WHERE P.RefP = D.RefP
AND D.RefCom = COM.RefCom
AND YEAR(COM.DateCom) = 2006
    -- psql : EXTRACT(YEAR FROM COM.DateCom) = 2006
ORDER BY P.TypeP, P.Prix DESC ;
```

```
+----------+--------+-------+
| TypeP    | Prix   | RefP  |
+----------+--------+-------+
| Cheville | 220.00 | CH464 |
| Cheville | 120.00 | CH264 |
| Clou     | 105.00 | CL45  |
| Clou     |  95.00 | CL60  |
| Planche  | 185.00 | PL224 |
+----------+--------+-------+
```

Tri de l'affichage : forme générale d'exécution

```
(3) SELECT [DISTINCT] A1, A2,..., Am (liste d'attributs
                                      et d'expressions calculées)
(1) FROM R1, R2,..., Rn (liste de relations)
(2) WHERE F  (expression de sélection)
(4) ORDER BY Ai [ASC|DESC],..., Aj [ASC|DESC] ;
```

Ordre d'exécution de la requête : (1) - (2) - (3) - (4)

Between

Noms des clients qui habitent Toulouse et dont la référence est inférieure à 6 ou comprise entre 11 et 15 :

```
SELECT C.NomC
FROM CLIENT C
WHERE C.Ville = 'Toulouse'
AND (RefC <= 6
OR RefC BETWEEN 11 AND 15) ;
```

```
+--------+
| NomC   |
+--------+
| GILLET |
| AVRON  |
| NEUMAN |
+--------+
```

Attention aux parenthèses :

```
SELECT C.NomC
FROM CLIENT C
WHERE C.Ville = 'Toulouse'
AND RefC <= 6
OR RefC BETWEEN 11 AND 15 ;
```

```
+-----------+
| NomC      |
+-----------+
| GILLET    |
| AVRON     |
| VANBIST   |
| NEUMAN    |
| FRANCK    |
| VANDERKA  |
| GUILLAUME |
+-----------+
```

Sans les parenthèses, tous les clients dont `RefC` est entre 11 et 15 sont dans le résultat.

LIKE

Filtrer une chaîne selon un *patron de motif* donné.
Deux caractères jokers :

- % : 0 ou plusieurs caractères
- _ : un caractère quelconque

Exemple : *Noms et villes des clients qui habitent dans une ville dont le nom (de la ville) se termine par un e :*

```
SELECT NomC, Ville
FROM CLIENT
WHERE Ville LIKE '%e'
ORDER BY Ville ;
```

NomC	Ville
MONTI	Geneve
VANBIST	Lille
GILLET	Toulouse
AVRON	Toulouse
MERCIER	Toulouse
PONCELET	Toulouse
NEUMAN	Toulouse

Noms et villes des clients qui habitent dans une ville dont le nom contient un e :

```
SELECT NomC, Ville
FROM CLIENT
WHERE Ville LIKE '%e%'
ORDER BY Ville ;
```

NomC	Ville
JACOB	Bruxelles
MONTI	Geneve
VANBIST	Lille
HANSENNE	Poitiers
FERARD	Poitiers
TOUSSAINT	Poitiers
GILLET	Toulouse
AVRON	Toulouse
MERCIER	Toulouse
PONCELET	Toulouse
NEUMAN	Toulouse

`Bruxelles` et `Poitiers` sont aussi dans le résultat.

Noms et villes des clients qui habitent dans une ville dont le nom contient au moins deux e :

```
SELECT NomC, Ville
FROM CLIENT
WHERE Ville LIKE '%e%e%'
ORDER BY Ville ;
```

```
+-------+-----------+
| NomC  | Ville     |
+-------+-----------+
| JACOB | Bruxelles |
| MONTI | Geneve    |
+-------+-----------+
```

Noms et villes des clients qui habitent dans une ville dont le deuxième caractère du nom de ville est un a :

```
SELECT NomC, Ville
FROM CLIENT
WHERE Ville LIKE '_a%'
ORDER BY Ville ;
```

```
+-----------+-------+
| NomC      | Ville |
+-----------+-------+
| GOFFIN    | Namur |
| FRANCK    | Namur |
| VANDERKA  | Namur |
| GUILLAUME | Paris |
+-----------+-------+
```

Expressions régulières (REGEXP [1])

Filtrer une chaîne selon un *motif* donné (la chaîne est dans le résultat dès que le motif est présent, peu importe ce qui est devant ou derrière le motif).

Caractères jokers :

- `.` un caractère
- `*`, `+`, `?` : répétition de ce qui précède
- `^` : ancrage début de chaîne
- `$` : ancrage fin de chaîne
- `[...]` : ensemble
- `|` : alternative
- `(...)` : atome de plusieurs caractères

[1] `postgres`: utilisez `~` au lieu de `REGEXP`

Noms et villes des clients qui habitent dans une ville dont le nom contient au moins deux e :

```
SELECT NomC, Ville
FROM CLIENT
WHERE Ville REGEXP 'e.*e'
ORDER BY Ville ;
```

```
+-------+-----------+
| NomC  | Ville     |
+-------+-----------+
| JACOB | Bruxelles |
| MONTI | Geneve    |
+-------+-----------+
```

(et pas '~~.*e.*e.*~~' : notez la différence par rapport à LIKE)

Noms et villes des clients qui habitent dans une ville dont le nom se termine par un e :

```
SELECT NomC, Ville
FROM CLIENT
WHERE Ville REGEXP 'e$'
ORDER BY Ville ;
```

```
+----------+----------+
| NomC     | Ville    |
+----------+----------+
| MONTI    | Geneve   |
| VANBIST  | Lille    |
| GILLET   | Toulouse |
| AVRON    | Toulouse |
| MERCIER  | Toulouse |
| PONCELET | Toulouse |
| NEUMAN   | Toulouse |
+----------+----------+
```

Pour chaque ville dont le nom contient la lettre 'e' ou la lettre 'i', les références des clients qui y habitent :

```
SELECT Ville, RefC
FROM CLIENT
WHERE Ville REGEXP '[ei]'
ORDER BY Ville ASC, RefC DESC ;
```

avec REGEXP

```
SELECT Ville, RefC
FROM CLIENT
WHERE Ville like '%e%'
OR Ville like '%i%'
ORDER BY Ville ASC, RefC DESC ;
```

avec LIKE

Calcul arithmétique

Détail des commandes en y incluant le type de chaque produit commandé, son prix unitaire et son prix total :

```
SELECT D.RefCom, D.RefP, P.TypeP, D.Quantite, P.Prix AS "Prix unitaire",
       P.Prix*D.Quantite AS PrixTotal
FROM DETAIL D, PRODUIT P
WHERE D.RefP=P.RefP
ORDER BY D.RefCom, PrixTotal DESC ;
```

RefCom	RefP	TypeP	Quantite	Prix unitaire	PrixTotal
1	CH464	Cheville	25	220.00	5500.00
2	CH262	Cheville	60	75.00	4500.00
2	CL60	Clou	20	95.00	1900.00
3	CL60	Clou	30	95.00	2850.00
4	CH464	Cheville	120	220.00	26400.00
4	CL45	Clou	20	105.00	2100.00
5	PL224	Planche	600	185.00	111000.00
5	CH464	Cheville	260	220.00	57200.00
5	CL60	Clou	15	95.00	1425.00
6	CL45	Clou	3	105.00	315.00
7	CH264	Cheville	180	120.00	21600.00
7	PL224	Planche	92	185.00	17020.00
7	CL60	Clou	70	95.00	6650.00
7	CL45	Clou	22	105.00	2310.00

calcul arithmétique : P.Prix*D.Quantite
renommage d'une colonne : AS (pour affichage noms colonnes et pour le ORDER BY)

Fonctions agrégats

- `COUNT` : comptage
- `SUM` : somme
- `AVG` : moyenne
- `MIN` : minimum
- `MAX` : maximum

Principe :
le calcul porte sur un ensemble de n-uplets (et non pas sur un seul n-uplet). Un tel ensemble est une table ou un élément d'une partition d'une table.

Fonctions agrégats

Nombre de produits, somme et moyenne des prix des produits, prix minimum et maximum des produits :

```
SELECT COUNT(*), SUM(Prix), AVG(Prix), Min(Prix), Max(Prix)
FROM PRODUIT ;
```

```
+----------+-----------+------------+-----------+-----------+
| COUNT(*) | SUM(Prix) | AVG(Prix)  | Min(Prix) | Max(Prix) |
+----------+-----------+------------+-----------+-----------+
|        7 |   1030.00 | 147.142857 |     75.00 |    230.00 |
+----------+-----------+------------+-----------+-----------+
```

`COUNT(*)` renvoie le nombre de lignes de la table `PRODUIT`

COUNT : exemples

```
SELECT COUNT(*), COUNT(Ville), COUNT(DISTINCT Ville)
FROM CLIENT ;
```

```
+----------+--------------+-----------------------+
| COUNT(*) | COUNT(Ville) | COUNT(DISTINCT Ville) |
+----------+--------------+-----------------------+
|       15 |           15 |                     7 |
+----------+--------------+-----------------------+
```

`COUNT(Ville)` : nombre de champs `Ville` non nuls.
`COUNT(DISTINCT Ville)` : nombre de champs `Ville` non nuls et distincts.

Partitionnement

Nombre de fois où le produit CL60 a été commandé :

```
SELECT D.RefP, COUNT(*)
FROM DETAIL D
WHERE D.RefP = 'CL60' ;
```

```
+------+----------+
| RefP | COUNT(*) |
+------+----------+
| CL60 |        4 |
+------+----------+
```

Nombre de fois où le produit CL45 a été commandé :

```
SELECT D.RefP, COUNT(*)
FROM DETAIL D
WHERE D.RefP = 'CL45' ;
```

```
+------+----------+
| RefP | COUNT(*) |
+------+----------+
| CL45 |        3 |
+------+----------+
```

Partitionnement

Pour chaque produit, nombre de fois où il a été commandé :
une requête pour chaque produit ?

- fastidieux...
- et la liste des produits n'est pas connue, sauf en interrogeant la base.

➥ utiliser un partitionnement avec `GROUP BY`

```
SELECT D.RefP, COUNT(*)
FROM DETAIL D
GROUP BY D.RefP ;
```

```
+-------+----------+
| RefP  | COUNT(*) |
+-------+----------+
| CH262 |        1 |
| CH264 |        1 |
| CH464 |        3 |
| CL45  |        3 |
| CL60  |        4 |
| PL224 |        2 |
+-------+----------+
```

Partitionnement

En ordonnant par ordre décroissant du nombre de ventes, puis ordre croissant suivant `RefP` :

```
SELECT D.RefP, COUNT(*) AS NB
FROM DETAIL D
GROUP BY D.RefP
ORDER BY NB DESC, D.RefP ;
```

```
+-------+----+
| RefP  | NB |
+-------+----+
| CL60  |  4 |
| CH464 |  3 |
| CL45  |  3 |
| PL224 |  2 |
| CH262 |  1 |
| CH264 |  1 |
+-------+----+
```

Partitionnement

Pour chaque produit de prix supérieur à 100, nombre de fois où il a été commandé :

➥ le `COUNT` doit porter sur un ensemble de n-uplets qui est l'ensemble des commandes d'un même produit : partitionnement avec `GROUP BY` selon `RefP`

```
SELECT D.RefP, COUNT(*)
FROM DETAIL D, PRODUIT P
WHERE D.RefP = P.RefP
AND P.Prix > 100
GROUP BY D.RefP ;
```

```
+-------+----------+
| RefP  | COUNT(*) |
+-------+----------+
| CH264 |        1 |
| CH464 |        3 |
| CL45  |        3 |
| PL224 |        2 |
+-------+----------+
```

Remarque :
la jointure élimine les produits non commandés (ici `PL222`).

Partitionnement

Pour chaque client qui a fait au moins une commande, le montant total de ses commandes :

➥ partitionnement selon les clients.

```
SELECT COM.RefC, SUM(P.Prix*D.Quantite)
FROM DETAIL D, COMMANDE COM, PRODUIT P
WHERE COM.RefCom=D.RefCom
AND D.RefP=P.RefP
GROUP BY COM.RefC ;
```

```
+------+-------------------------+
| RefC | SUM(P.Prix*D.Quantite) |
+------+-------------------------+
|    7 |                47580.00 |
|    9 |                35215.00 |
|   12 |               169625.00 |
|   14 |                 8350.00 |   -> 8350 = 25*220 + 30*95
+------+-------------------------+
```

Partitionnement avec sélection

Pour chaque client qui a fait au moins une commande, le montant total de ses commandes lorsque les commandes d'un client ont un montant supérieur à 10000 :

➥ sélection sur les éléments de la partition : `HAVING`

```
SELECT COM.RefC, SUM(P.Prix*D.Quantite)
FROM DETAIL D, COMMANDE COM, PRODUIT P
WHERE COM.RefCom=D.RefCom
AND D.RefP=P.RefP
GROUP BY COM.RefC
HAVING SUM(P.Prix*D.Quantite) > 10000 ;
```

```
+------+-------------------------+
| RefC | SUM(P.Prix*D.Quantite) |
+------+-------------------------+
|    7 |                47580.00 |
|    9 |                35215.00 |
|   12 |               169625.00 |
+------+-------------------------+
```

Partitionnement avec sélection

Pour chaque client qui a fait au moins une commande, le montant total de ses commandes lorsque les commandes d'un client ont un montant supérieur à 10000 et que le client a commandé strictement moins de 5 produits :

```
SELECT COM.RefC, SUM(P.Prix*D.Quantite)
FROM DETAIL D, COMMANDE COM, PRODUIT P
WHERE COM.RefCom=D.RefCom
AND D.RefP=P.RefP
GROUP BY COM.RefC
HAVING (SUM(P.Prix*D.Quantite) > 10000
AND COUNT(D.RefP) < 5)
```

RefC	SUM(P.Prix*D.Quantite)
7	47580.00
12	169625.00

Partitionnement avec sélection

Rappel : par défaut, SQL n'élimine pas les doublons. Si on introduit DISTINCT :

```
SELECT COM.RefC, SUM(P.Prix*D.Quantite)
FROM DETAIL D, COMMANDE COM, PRODUIT P
WHERE COM.RefCom=D.RefCom
AND D.RefP=P.RefP
GROUP BY COM.RefC
HAVING (SUM(P.Prix*D.Quantite) > 10000
AND COUNT(DISTINCT D.RefP) < 5)
```

```
+------+-------------------------+
| RefC | SUM(P.Prix*D.Quantite) |
+------+-------------------------+
|    7 |                47580.00 |
|    9 |                35215.00 |
|   12 |               169625.00 |
+------+-------------------------+
```

Le client dont RefC est égal à 9 est alors dans le résultat : il a commandé 5 produits, mais 4 produits distincts.

Forme générale d'une requête SQL

```
(5) SELECT [DISTINCT] A1, A2,..., Am (liste d'attributs
                                      et d'expressions calculées)
(1) FROM R1, R2,..., Rn (liste de relations)
(2) WHERE F  (expression de sélection)
(3) GROUP BY (clé de partitionnement
(4) HAVING (selection sur un groupe)
(6) ORDER BY Ai [ASC|DESC],..., Aj [ASC|DESC] ;
```

Ordre d'exécution de la requête : (1) - (2) - (3) - (4) - (5) - (6)

On ne peut pas avoir de `HAVING` sans `GROUP BY` (puisque le `HAVING` sélectionne des groupes).

Requêtes imbriquées

Pour exprimer des notions comme :

- la non appartenance
- les objets maximaux/minimaux
- calcul avec des granularités différentes d'ensembles (comparaison, calcul de pourcentage)
- division de l'algèbre relationnelle
- ...

Requêtes imbriquées : appartenance

Références de produits qui n'ont jamais été commandés :

```
SELECT RefP
FROM PRODUIT
WHERE RefP NOT IN
      (SELECT RefP
       FROM DETAIL) ;
```

`IN` : signifie $\in$
`NOT IN` : signifie $\notin$

L'élément dont l'appartenance est testée (ici `RefP`) doit être du type des éléments retournés par le `SELECT` imbriqué.

Requêtes imbriquées : appartenance

Références et prix des produits de prix supérieur à 100 et qui ont été commandés au moins deux fois en 2005 :

```
SELECT RefP, Prix
FROM PRODUIT
WHERE Prix > 100
AND RefP IN
  (SELECT RefP
   FROM DETAIL D, COMMANDE COM
   WHERE D.RefCom = COM.RefCom
   AND YEAR(DateCom) = '2005'
   GROUP BY RefP
   HAVING COUNT(*) >= 2) ;
```

```
+-------+--------+
| RefP  | Prix   |
+-------+--------+
| CH464 | 220.00 |
+-------+--------+
```

L'élément dont l'appartenance est testée (ici `RefP`) doit être du type des éléments retournés par le `SELECT` imbriqué.

La granularité du `SELECT` interne (groupe de produits) est différente de celle du `SELECT` externe (un produit).

Requêtes imbriquées : appartenance

Références et prix des produits de prix supérieur à 100 et qui ont été commandés en 2005 :

une requête imbriquée n'est pas nécessaire :

```
SELECT D.RefP, P.Prix
FROM PRODUIT P, DETAIL D, COMMANDE COM
WHERE P.RefP = D.RefP
AND D.RefCom = COM.RefCom
AND P.Prix > 100
AND YEAR(COM.DateCom) = '2005' ;
```

```
+-------+--------+
| RefP  | Prix   |
+-------+--------+
| CH464 | 220.00 |
| CH464 | 220.00 |
| CL45  | 105.00 |
+-------+--------+
```

Requêtes imbriquées : appartenance

La requête *'Références et prix des produits de prix supérieur à 100 et qui ont été commandés en 2005"* peut aussi être écrite avec une requête imbriquée mais au prix d'une forte dégradation de la lisibilité et de la déclarativité. L'écriture suivante est donc déconseillée :

```
SELECT RefP, Prix
FROM PRODUIT
WHERE Prix > 100
AND RefP IN
      (SELECT RefP
       FROM DETAIL D, COMMANDE COM
       WHERE D.RefCom = COM.RefCom
       AND YEAR(DateCom) = '2005') ;
```

Requêtes imbriquées : appartenance

Pour chaque produit dont le prix est supérieur à 100, références des clients n'ayant jamais acheté ce produit :

```
SELECT DISTINCT  P.RefP, COM.RefC
FROM PRODUIT P, COMMANDE COM
WHERE P.Prix >= 100
AND (P.RefP, COM.RefC) NOT IN
      (SELECT D.RefP, COM.RefC
       FROM DETAIL D, COMMANDE COM
       WHERE COM.RefCom = D.RefCom) ;
```

Remarques :

- produit cartésien pour générer toutes les paires (produit, client)
- le test d'appartenance s'effectue sur une liste d'attributs
- en utilisant `COMMANDE` dans le premier `SELECT`, on considère uniquement les clients qui ont commandé au moins une fois.
  Pour considérer tous les clients : utiliser `CLIENT`

Requêtes imbriquées

Produit le plus cher (1/2)

```
SELECT MAX(Prix)
FROM PRODUIT ;
```

Référence et prix du produit le plus cher : si on écrit :

```
SELECT RefP, MAX(Prix)
FROM PRODUIT ;
```

Le résultat est :

```
ERROR 1140 (42000): In aggregated query without GROUP BY, expression #1 of
SELECT list contains nonaggregated column 'cremilleux_bd.PRODUIT.RefP';
this is incompatible with sql_mode=only_full_group_by
```

`RefP` n'est pas unique sur la granularité sur laquelle l'agrégat est calculé (ici, la table `PRODUIT`)

Requêtes imbriquées

Produit le plus cher (2/2)

Un SELECT imbriqué est nécessaire :

```
SELECT RefP, Prix
FROM PRODUIT
WHERE Prix =
   (SELECT MAX(Prix)
    FROM PRODUIT) ;
```

```
+-------+--------+
| RefP  | Prix   |
+-------+--------+
| PL222 | 230.00 |
+-------+--------+
```

Remarque : le maximum n'est pas forcément unique.

pour cet exemple, >= ALL peut aussi être utilisé.

ALL : comparaison ensembliste.

```
SELECT RefP, Prix
FROM PRODUIT
WHERE Prix >= ALL
   (SELECT MAX(Prix)
    FROM PRODUIT) ;
```

Requêtes imbriquées : comparaison

Références et prix des produits de type cheville qui sont moins chers que le produit référencé PL224 :

```
SELECT RefP, Prix
FROM PRODUIT
WHERE TypeP = "Cheville"
and Prix < (SELECT Prix
            FROM PRODUIT
            WHERE RefP = "PL224") ;
```

Pour que la comparaison soit correcte, la requête imbriquée doit ici retourner une valeur simple (i.e. une table à 1 ligne et 1 colonne).

Requêtes imbriquées : calcul de pourcentage

Nombre de produits achetés par client :

```
SELECT COM.RefC, COUNT(DISTINCT D.RefP)
FROM DETAIL D, COMMANDE COM
WHERE COM.RefCom=D.RefCom
GROUP BY COM.RefC ;
```

Pourcentage de produits achetés par client :

```
SELECT COM.RefC, (COUNT(DISTINCT D.RefP)*100/(SELECT COUNT(*)
                                    FROM PRODUIT)) AS Pourcentage
FROM DETAIL D, COMMANDE COM
WHERE COM.RefCom=D.RefCom
GROUP BY COM.RefC
ORDER BY Pourcentage DESC ;
```

Remarquez le SELECT imbriqué (i.e. `(SELECT COUNT(*) FROM PRODUIT)`) pour calculer le nombre total de produits.

Requêtes imbriquées dans le FROM (1/2)

Les deux calculs (sur des granularités différentes) sont effectués par deux requêtes imbriquées dans le FROM.

```
SELECT PRODUIT_CLIENT.RefC, (NbProduitsParClient*100)/NbProduitsTotal AS
                                                              Pourcentage
FROM (SELECT COM.RefC, COUNT(DISTINCT D.RefP) AS NbProduitsParClient
      FROM DETAIL D, COMMANDE COM
      WHERE COM.RefCom = D.RefCom
      GROUP BY COM.RefC) PRODUIT_CLIENT,
      (SELECT COUNT(*) AS NbProduitsTotal
      FROM PRODUIT) NB_PRODUIT
ORDER BY Pourcentage DESC ;
```

```
+------+-------------+
| RefC | Pourcentage |
+------+-------------+
|    7 |     57.1429 |
|    9 |     57.1429 |
|   12 |     42.8571 |
|   14 |     28.5714 |
+------+-------------+
```

Requêtes imbriquées dans le FROM (2/2)

Ce qui est faux : l'erreur provient du fait que SQL n'a pas de garantie que NB_PRODUIT.NbProduitsTotal a une valeur unique pour chaque élément du GROUP BY

```
SELECT COM.RefC, COUNT(DISTINCT D.RefP)*100/NB_PRODUIT.NbProduitsTotal AS
                                                              Pourcentage
FROM (SELECT COUNT(*) AS NbProduitsTotal
      FROM PRODUIT) NB_PRODUIT,
      DETAIL D, COMMANDE COM
WHERE COM.RefCom=D.RefCom
GROUP BY COM.RefC
ORDER BY Pourcentage DESC ;

ERROR 1055 (42000): Expression #2 of SELECT list is not in GROUP BY clause
and contains nonaggregated column 'NB_PRODUIT.NbProduitsTotal' which is not
functionally dependent on columns in GROUP BY clause; this is incompatible
with sql_mode=only_full_group_by
```

NOT EXISTS

Références des clients qui n'ont pas passé de commande :

Reformulation : référence de chaque client C tel qu'il n'existe pas de commande COM faite par C

```
SELECT C.NomC
FROM CLIENT C
WHERE NOT EXISTS
  (SELECT *
   FROM COMMANDE COM
   WHERE COM.RefC = C.RefC) ;
```

La requête imbriquée est ici corrélée avec la requête principale :
le `SELECT` imbriqué est exécuté pour chaque valeur de la variable `CLIENT C` déclarée à l'extérieur.

NOT EXISTS

Cette requête (rappel : *Références des clients qui n'ont pas passé de commande*) peut aussi être écrite avec `NOT IN`
(et cette écriture est certainement plus naturelle)

```
SELECT NomC
FROM CLIENT
WHERE RefC NOT IN (SELECT RefC
                   FROM COMMANDE) ;
```

La sous-requête (non corrélée) est exécutée une seule fois.

NATURAL JOIN : pour la jointure naturelle

Quantité totale de chevilles commandées par des clients de Toulouse :

```
SELECT SUM(Quantite)
FROM CLIENT, COMMANDE, DETAIL, PRODUIT
WHERE CLIENT.RefC = COMMANDE.RefC
AND COMMANDE.RefCom = DETAIL.RefCom
AND DETAIL.RefP = PRODUIT.RefP
AND TypeP = 'Cheville'
AND Ville = 'Toulouse' ;
```

Avec `NATURAL JOIN` :

```
SELECT SUM(Quantite)
FROM CLIENT NATURAL JOIN COMMANDE NATURAL JOIN DETAIL NATURAL JOIN PRODUIT
WHERE TypeP = 'Cheville'
AND Ville = 'Toulouse' ;
```

ATTENTION : `NATURAL JOIN` effectue la jointure sur tous les attributs de mêmes noms entre les tables sans autre considération

(dans la pratique, une jointure doit s'effectuer sur des attributs qui ont une signification qui "correspondent" pour que le résultat ait du sens).

NATURAL JOIN : pour la jointure naturelle

Ajout d'une table COMMUNE :

```
CREATE TABLE COMMUNE (
   Commune VARCHAR(20) NOT NULL,
   Pays VARCHAR(20) NOT NULL,
   Departement INT NOT NULL,
   PRIMARY KEY (Commune)
) ;
```

Namur	Belgique	99
Poitiers	France	86
Geneve	Suisse	99
Toulouse	France	31
Bruxelles	Belgique	99
Paris	France	75
Rome	Italie	99
Londres	Grande-Bretagne	99
Lyon	France	69
Anvers	Belgique	99
Lille	France	59

Noms et pays des clients : quel est le résultat de la requête ?

```
SELECT CLIENT.NomC, COMMUNE.Pays
FROM CLIENT NATURAL JOIN COMMUNE ;
```

NATURAL JOIN : pour la jointure naturelle

Ajout d'une table COMMUNE :

```
CREATE TABLE COMMUNE (
    Commune VARCHAR(20) NOT NULL,
    Pays VARCHAR(20) NOT NULL,
    Departement INT NOT NULL,
    PRIMARY KEY (Commune)
) ;
```

Namur	Belgique	99
Poitiers	France	86
Geneve	Suisse	99
Toulouse	France	31
Bruxelles	Belgique	99
Paris	France	75
Rome	Italie	99
Londres	Grande-Bretagne	99
Lyon	France	69
Anvers	Belgique	99
Lille	France	59

Noms et pays des clients : quel est le résultat de la requête ?

```
SELECT CLIENT.NomC, COMMUNE.Pays
FROM CLIENT NATURAL JOIN COMMUNE ;
```

Dans cet exemple, comme il n'y a pas d'attribut commun entre les deux tables, cela revient à un produit cartésien (ici : 165 n-uplets).
Il n'y a pas d'erreur de syntaxe.

NATURAL JOIN : pour la jointure naturelle

Pour que cette requête soit correcte, il faut expliciter la jointure entre CLIENT et COMMUNE :

```
SELECT C.NomC, CO.Pays
FROM CLIENT C, COMMUNE CO
WHERE C.Ville = CO.Commune ;
```

```
+-----------+----------+
| NomC      | Pays     |
+-----------+----------+
| GOFFIN    | Belgique |
| HANSENNE  | France   |
| MONTI     | Suisse   |
| GILLET    | France   |
| AVRON     | France   |
| FERARD    | France   |
| MERCIER   | France   |
| TOUSSAINT | France   |
| PONCELET  | France   |
| JACOB     | Belgique |
| VANBIST   | France   |
| NEUMAN    | France   |
| FRANCK    | Belgique |
| VANDERKA  | Belgique |
| GUILLAUME | France   |
+-----------+----------+
```

NATURAL JOIN : pour la jointure naturelle

Un nom de commune n'est pas forcément unique : ajout d'un identifiant RefC, pour une capitale, la valeur de RefC est une dizaine (cf. table COMMUNE2) :

```
CREATE TABLE COMMUNE2(
   RefC INT NOT NULL,
   Commune VARCHAR(20) NOT NULL,
   Pays VARCHAR(20) NOT NULL,
   Departement INT NOT NULL,
   PRIMARY KEY (RefC)
) ;
```

12	Namur	Belgique	99
22	Poitiers	France	86
50	Geneve	Suisse	99
21	Toulouse	France	31
10	Bruxelles	Belgique	99
20	Paris	France	75
40	Rome	Italie	99
30	Londres	Grande-Bretagne	99
25	Lyon	France	69
11	Anvers	Belgique	99
23	Lille	France	59

NATURAL JOIN : pour la jointure naturelle

Quel est le résultat de la requête ?

```
SELECT *
FROM CLIENT NATURAL JOIN COMMUNE2 ;
```

NATURAL JOIN : pour la jointure naturelle

Quel est le résultat de la requête ?

```
SELECT *
FROM CLIENT NATURAL JOIN COMMUNE2 ;
```

RefC	NomC	Ville	CAT	Commune	Pays	Departement
10	JACOB	Bruxelles	C2	Bruxelles	Belgique	99
11	VANBIST	Lille	B1	Anvers	Belgique	99
12	NEUMAN	Toulouse	C2	Namur	Belgique	99

RefC est un attribut de même nom pour les 2 tables : NATURAL JOIN le considère pour la jointure même si les sémantiques de ces deux attributs sont distinctes. Le résultat pour JACOB est correct uniquement par hasard !

Remarque : "Noms et pays des clients" : même requête que précédemment :

```
SELECT C.NomC, CO.Pays
FROM CLIENT C, COMMUNE2 CO
WHERE C.Ville = CO.Commune ;
```

La division entre ensembles en algèbre relationnelle

Noms des personnes qui ont visité *toutes* les capitales (on suppose ici qu'une personne voyageant dans un pays visite la capitale de ce pays).

Si on suppose que Voyage contient :

nompers	nompays
chloe	macedoine
chloe	irlande
mehdi	irlande
mehdi	perou
helena	perou
chloe	autriche
chloe	perou

Rappel : Pays[capitale] : `dublin, vienne, lima, skopje`

Voyage ⋈ Pays

nompers	nompays	capitale
chloe	macedoine	skopje
chloe	irlande	dublin
mehdi	irlande	dublin
mehdi	perou	lima
helena	perou	lima
chloe	autriche	vienne
chloe	perou	lima

la requête `(Voyage ⋈ Pays)[nompers, capitale] / Pays[capitale]` produit le résultat.

Division entre ensembles en SQL (1/2)

Références des clients qui ont acheté au moins un produit de chaque type de produits (i.e., ont acheté tous les types de produits).

Écriture de la division selon les opérateurs fondamentaux de l'algèbre relationnelle : (utilisation d'une double différence) :

Soient $R(X_1, ..., X_r, X_{r+1}, ..., X_m)$ et $S(Y_{r+1}, ..., Y_m)$

alors $R/S = R[X_1, ..., X_r] - ((R[X_1, ..., X_r] \times S) - R)[X_1, ..., X_r]$

Sur notre exemple :

$\texttt{COM[RefC]-(((COM}\times\texttt{P)[COM.RefC,P.TypeP]) -}$
$\texttt{((COM}\bowtie\texttt{D}\bowtie\texttt{P)[COM.RefC,P.TypeP]))[RefC]}$

```
SELECT COM.RefC
FROM COMMANDE COM
WHERE COM.RefC NOT IN
    (SELECT COM.RefC
     FROM COMMANDE COM, PRODUIT P
     WHERE (COM.RefC, P.TypeP) NOT IN
         (SELECT COM.RefC, P.TypeP
          FROM COMMANDE COM, DETAIL D, PRODUIT P
          WHERE COM.RefCom = D.RefCom
          AND D.RefP = P.RefP)) ;
```

Division entre ensembles en SQL (2/2)

Avec comptage : clients dont le nombre de types de produits commandés est égal au nombre de types de produits dans la base.

```
SELECT COM.RefC
FROM COMMANDE COM, DETAIL D, PRODUIT P
WHERE COM.RefCom = D.RefCom
AND D.RefP = P.RefP
GROUP BY COM.RefC
HAVING COUNT(DISTINCT P.TypeP) =
      (SELECT COUNT(DISTINCT P.TypeP)
       FROM PRODUIT P) ;
```

`DISTINCT` est indispensable pour calculer le nombre correct de types de produits.

Pour obtenir les noms des clients (au lieu de leurs références) :
(écriture avec une requête imbriquée dans un FROM)

```
SELECT C.NomC
FROM (SELECT COM.RefC
      FROM COMMANDE COM
      WHERE COM.RefC NOT IN
      (SELECT COM.RefC
       FROM COMMANDE COM, PRODUIT P
       WHERE (COM.RefC, P.TypeP) NOT IN
           (SELECT COM.RefC, P.TypeP
            FROM COMMANDE COM, DETAIL D, PRODUIT P
            WHERE COM.RefCom = D.RefCom
            AND D.RefP = P.RefP))) CLIENT_TOUS_TYPEP,
      CLIENT C
WHERE CLIENT_TOUS_TYPEP.RefC = C.RefC ;
```

```
+---------+
| NomC    |
+---------+
| NEUMAN  |
| MERCIER |
+---------+
```

En utilisant CLIENT dans le premier SELECT, il n'est plus nécessaire d'imbriquer une requête :

```
SELECT C.NomC
FROM CLIENT C, COMMANDE COM
WHERE C.RefC = COM.RefC
AND C.RefC NOT IN
    (SELECT COM.RefC
     FROM COMMANDE COM, PRODUIT P
     WHERE (COM.RefC, P.TypeP) NOT IN
         (SELECT COM.RefC, P.TypeP
          FROM COMMANDE COM, DETAIL D, PRODUIT P
          WHERE COM.RefCom = D.RefCom
          AND D.RefP = P.RefP)) ;
```

```
+---------+
| NomC    |
+---------+
| NEUMAN  |
| MERCIER |
+---------+
```

Question : dans la requête précédente, pourquoi la jointure avec COMMANDE du premier SELECT est nécessaire ?

Autrement dit, pourquoi la requête ci-dessous ne produit pas le résultat demandé ?

```
SELECT C.NomC
FROM CLIENT C
WHERE C.NomC NOT IN
    (SELECT C.NomC
     FROM COMMANDE COM, PRODUIT P, CLIENT C
     WHERE C.RefC = COM.RefC
     AND (COM.RefC, P.TypeP) NOT IN
         (SELECT COM.RefC, P.TypeP
          FROM COMMANDE COM, DETAIL D, PRODUIT P
          WHERE COM.RefCom = D.RefCom
          AND D.RefP = P.RefP)) ;
```

Autres

- **contraintes :** clé étrangère : dans la table `DETAIL`

  `FOREIGN KEY (RefP) REFERENCES PRODUIT`

  impose que toute valeur de RefP dans une ligne de la table DETAIL soit présente comme identifiant primaire (`PRIMARY KEY`) de la table `PRODUIT`
- `DROP TABLE IF EXISTS CLIENT, PRODUIT, DETAIL, COMMANDE ;`
- `INSERT`
- `DELETE FROM <NomTable> WHERE <expression de sélection>`
- `UPDATE <Relation> SET <Liste affectations attributs> WHERE <expression de sélection>`

Remerciements

Ces diapositives, et notamment les exemples, doivent beaucoup à Etienne Grandjean et Jean-Jacques Hébrard, enseignants-chercheurs au département mathématiques et informatique de l'Université de Caen Normandie.