

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).

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.

3/66

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.  
[...]  
mysql>
```

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

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

4/66

La première fois, il faut créer sa base :

sous mysql :

```
mysql> CREATE DATABASE LOGIN_bd ;
```

puis se connecter à sa base :

```
mysql> 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
```

5/66

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)

6/66

Liste des tables : mysql> **show tables** ;

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

Schéma d'une table : mysql> **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    |       |
+-----+-----+-----+-----+-----+-----+
```

7/66

Contenu d'une table : mysql> **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 : /* ... */

8/66

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.

9/66

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>

10/66

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 [...] ;
```

11/66

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' ;
```

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

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

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

12/66

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
```

13/66

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 :

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

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   |  
+-----+-----+-----+-----+
```

14/66

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

Calcul des n-uplets :

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

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 : $(C \times COM \times D) : ((C.RefC = COM.RefC) \wedge$
 $(COM.RefCom = D.RefCom)) [NomC, RefP]$

Produit cartésien - sélection - projection

15/66

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 :

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

Le SELECT correspond à une projection


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

Les tables ne sont pas forcément distinctes.

Exemple : paires de noms de clients habitant 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 :

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

17/66

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  |  
+-----+-----+-----+
```

18/66

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)

19/66

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.

20/66

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

21/66

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
+	-----+	-----+

23/66

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

¹postgres: 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	
+-----+-----+		

25/66

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**

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)

27/66

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.

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

29/66

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.

30/66

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

31/66

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

32/66

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

33/66

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).

34/66

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

35/66

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

36/66

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

37/66

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.

38/66

```
(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).