

# Recherche des données

Pierre Lefebvre

# Introduction

- Le sous-langage DML de SQL permet de consulter le contenu des tables et de les modifier. Il comporte 4 verbes.
- La requête **select** extrait des données des tables
- La requête **insert** insère de nouvelles lignes dans une table
- La requête **delete** supprime des lignes d'une table
- La requête **update** modifie les valeurs de colonnes de lignes existantes

# Sélectionner des enregistrements

- Pour extraire de votre base de données des informations, comme la liste des personnes de votre carnet d'adresse qui vivent à Paris.

- Syntaxe générale :

```
SELECT [ DISTINCT ] attributs  
      [ INTO OUTFILE fichier ]  
      [ FROM relation ]  
      [ WHERE condition ]  
      [ GROUP BY attributs [ ASC | DESC ] ]  
      [ HAVING condition ]  
      [ ORDER BY attributs ]  
      [ LIMIT [a,] b ]
```

- Exemple :

```
•      SELECT nom,prénom FROM Personnes WHERE adresse LIKE  
          '%paris%'
```

# Sélectionner des enregistrements

Nom	Description
<b>SELECT</b>	Spécifie les attributs dont on souhaite connaître les valeurs.
<b>DISTINCT</b>	Permet d'ignorer les doublons de ligne de résultat.
<b>INTO OUTFILE</b>	Spécifie le fichier sur lequel effectuer la sélection.
<b>FROM</b>	Spécifie le ou les relations sur lesquelles effectuer la sélection.
<b>WHERE</b>	Définie le ou les critères de sélection sur des attributs.
<b>GROUP BY</b>	Permet de grouper les lignes de résultats selon un ou des attributs.
<b>HAVING</b>	Définie un ou des critères de sélection sur des ensembles de valeurs d'attributs après groupement.
<b>ORDER BY</b>	Permet de définir l'ordre ( <b>ASC</b> endant par défaut ou <b>DESC</b> endant) dans l'envoi des résultats.
<b>LIMIT</b>	Permet de limiter le nombre de lignes du résultats

# Exemples

- Procérons par étapes :
- Pour sélectionner tous les enregistrements d'une relation :  
**SELECT \* FROM *relation***
- Pour sélectionner toutes les valeurs d'un seul attribut :  
**SELECT *attribut* FROM *relation***
- Pour éliminer les doublons :  
**SELECT DISTINCT *attribut* FROM *relation***
- Pour trier les valeurs en ordre croissant :  
**SELECT DISTINCT *attribut* FROM *relation* ORDER BY *attribut* ASC**
- Pour se limiter aux **num** premiers résultats :  
**SELECT DISTINCT *attribut* FROM *relation* ORDER BY *attribut* ASC  
LIMIT num**
- Pour ne sélectionner que ceux qui satisfont à une condition :  
**SELECT DISTINCT *attribut* FROM *relation* WHERE *condition*  
ORDER BY *attribut* ASC LIMIT num**

# Sélectionner des enregistrements

- Relation de départ :
- **SELECT \* FROM Gens**

1

<b>Gens</b>		
<i>Nom</i>	<i>Prenom</i>	<i>Age</i>
Dupond	Pierre	24
Martin	Marc	48
Dupont	Jean	51
Martin	Paul	36
Dupond	Lionel	68
Chirac	Jacques	70

- **SELECT *Nom* FROM Gens**
- **SELECT DISTINCT *Nom* FROM Gens**

2

<b>Gens</b>
<i>Nom</i>
Dupond
Martin
Dupont
Martin
Dupond
Chirac

3

<b>Gens</b>
<i>Nom</i>
Dupond
Martin
Dupont
Chirac

# Sélectionner des enregistrements

- SELECT DISTINCT  
*Nom* FROM *Gens*  
ORDER BY *Nom* ASC

<i>Gens</i>
<i>Nom</i>
Chirac
Dupond
Dupont
Martin

5

- SELECT DISTINCT  
*Nom* FROM *Gens*  
ORDER BY *Nom* ASC  
LIMIT 2

<i>Gens</i>
<i>Nom</i>
Chirac
Dupond

6

- SELECT DISTINCT *Nom* FROM *Gens* WHERE *Nom* <> 'Chirac' ORDER BY *Nom* ASC LIMIT 2

<i>Gens</i>
<i>Nom</i>
Dupond

# Extraction simple : projection

```
select NCLI, NOM, LOCALITE  
from CLIENT;
```

NCLI	NOM	LOCALITE
B062	GOFFIN	Namur
B112	HANSENNE	Poitiers
B332	MONTI	Genève
B512	GILLET	Toulouse
C003	AVRON	Toulouse
C123	MERCIER	Namur
C400	FERARD	Poitiers
D063	MERCIER	Toulouse
F010	TOUSSAINT	Poitiers
F011	PONCELET	Toulouse
F400	JACOB	Bruxelles
K111	VANBIST	Lille
K729	NEUMAN	Toulouse
L422	FRANCK	Namur
S127	VANDERKA	Namur
S712	GUILLAUME	Paris

```
select *  
from CLIENT;
```

\* = liste des colonnes

# Extraction simple: restriction

```
SELECT [ { DISTINCT | DISTINCTROW } | ALL ] listeColonnes  
FROM nomTable [aliasTable]  
[ WHERE condition ] ;
```

```
select NCLI, NOM  
from CLIENT  
where LOCALITE = 'Toulouse' ;
```

NCLI	NOM
B512	GILLET
C003	AVRON
D063	MERCIER
F011	PONCELET
K729	NEUMAN

# Extraction simple : doublons

```
select LOCALITE  
from CLIENT  
where CAT = 'C1';
```

LOCALITE
Poitiers
Namur
Poitiers
Namur
Namur

lignes en double

```
select distinct LOCALITE  
from CLIENT  
where CAT = 'C1';
```

LOCALITE
Namur
Poitiers

# Conditions plus complexes: les valeurs *null*

```
select NCLI  
from   CLIENT  
where  CAT = null;
```

***null* ne peut être comparé à rien,  
même pas à lui-même !**

NCLI

```
select NCLI  
from   CLIENT  
where  CAT is null;
```

NCLI
D063
K729

```
select NCLI  
from   CLIENT  
where  CAT is not null;
```

# *In et between*

```
select NCLI  
from CLIENT  
where CAT in ('C1','C2','C3');
```

```
select NCLI  
from CLIENT  
where LOCALITE not in ('Toulouse','Breda');
```

```
select NCLI  
from CLIENT  
where COMPTE between 1000 and 4000;
```

# Les masques

```
select NCLI  
from CLIENT  
where CAT like 'B_';
```

'\_' = un caractère quelconque

```
select NPRO  
from PRODUIT  
where LIBELLE like '%SAPIN%';
```

'%' = une chaîne quelconque

masques

Un masque définit une famille de chaînes de caractères :

'B\_'  
→  
'B1'  
'Bd'  
'B '

'%SAPIN%'  
→  
'PL. SAPIN 200x20x2'  
'Boite en SAPIN'  
'SAPIN VERNI'

'B\_'  
→  
'xB'  
'B'  
'B12'

'%SAPIN%'  
→  
'Boite en Sapin'  
'Achetez S A P I N !'

# Combinaisons logiques

```
select NOM, ADRESSE, COMPTE  
from CLIENT  
where LOCALITE = 'Toulouse' and COMPTE < 0;
```

```
select NOM, ADRESSE, COMPTE  
from CLIENT  
where COMPTE > 0  
and (CAT = 'C1' or LOCALITE = 'Paris')
```

# Données extraites et données dérivées

```
select 'TVA de ', NPRO, ' = ', 0.21*PRIXT*QSTOCK  
from PRODUIT  
where QSTOCK > 500;
```

TVA de	NPRO	=	0,21*PRIXT*QSTOCK
TVA de	CS264	=	67788
TVA de	PA45	=	12789
TVA de	PH222	=	37770.6
TVA de	PS222	=	47397

```
select NPRO as Produit, 0.21*PRIXT*QSTOCK as Valeur_TVA  
from PRODUIT  
where QSTOCK > 500;
```

"Produit" est  
un alias de colonne

Produit	Valeur_TVA
CS264	67788
PA45	12789
PH222	37770.6
PS222	47397

# Les fonctions

# Les fonctions pour les caractères

Fonction	Objectif	Exemple			
ASCII( <i>c</i> )	Retourne le caractère ASCII équivalent.	<code>ASCII('A')</code> donne 65			
CHAR( <i>n</i> )	Retourne le caractère équivalent dans le jeu de caractères en cours.	<code>CHR(161)    CHR(162)</code> donne îô			
CONCAT( <i>c1,c2</i> )	Concatène deux chaînes.	<pre>SELECT CONCAT(     CONCAT(nom, ' vole pour '), compa)     "Personnel" FROM Pilote;</pre> <table border="1"> <tr><td>Personnel</td></tr> <tr><td>viel travaille pour AF</td></tr> <tr><td>...</td></tr> </table>	Personnel	viel travaille pour AF	...
Personnel					
viel travaille pour AF					
...					
FIELD( <i>c,c1,c2...</i> )	Retourne l'index qui correspond à la première égalité entre <i>c</i> et <i>c1</i> , <i>c</i> et <i>c2</i> , etc. 0 si aucune égalité n'est trouvée.	<pre>SELECT FIELD('Air', 'air', 'Airbus', 'Air') "Attention à la casse!";</pre> <table border="1"> <tr><td>Attention à la casse!</td></tr> <tr><td>1</td></tr> </table>	Attention à la casse!	1	
Attention à la casse!					
1					
INSERT( <i>c1,pos,t,c2</i> )	Modifie la chaîne <i>c1</i> en insérant <i>t</i> caractères de la sous-chaîne <i>c2</i> à partir de la position <i>pos</i> .	<pre>SELECT INSERT('Compxxxie : Airbus ',5, 3, 'agn') "Qui?";</pre> <table border="1"> <tr><td>Qui?</td></tr> <tr><td>Compagnie : Airbus</td></tr> </table>	Qui?	Compagnie : Airbus	
Qui?					
Compagnie : Airbus					
INSTR( <i>c1,c2</i> )	Premier indice d'une sous-chaîne <i>c1</i> dans une chaîne <i>c2</i> . Exemple : indice de 'Air' dans la chaîne.	<pre>SELECT INSTR('Infos-Air : AirBus pour Air-France','Air') "Indice";</pre> <table border="1"> <tr><td>Indice</td></tr> <tr><td>7</td></tr> </table>	Indice	7	
Indice					
7					

# Les fonctions aggregatives

```
select 'Namur',avg(COMPTE) as Moyenne,  
       max(COMPTE)-min(COMPTE) as Ecart_max,  
       count(*) as Nombre  
  from CLIENT  
 where LOCALITE = 'Namur' ;
```

Namur	Moyenne	Ecart_max	Nombre
Namur	-2520	4580	4

le résultat ne comprend  
qu'une seule ligne

```
select sum(QSTOCK*PRIX)  
  from PRODUIT  
 where LIBELLE like '%SAPIN%' ;
```

# Les fonctions aggregatives

*Attention aux valeurs dupliquées*

```
select count(NCLI)
from   COMMANDE;
```

count(NCLI)
7

```
select distinct count(NCLI)
from   COMMANDE;
```

count(NCLI)
7

```
select count(distinct NCLI)
from   COMMANDE;
```

count(NCLI)
5

# Les fonctions aggregatives

```
select count(NCLI) as Numeros,  
       count(NOM)  as Noms,  
       count(LOCALITE) as Localites,  
       count(CAT)   as Categories  
  from CLIENT;
```

Numeros	Noms	Localites	Categories
16	16	16	14

```
select count(distinct NCLI) as Numeros,  
       count(distinct NOM)  as Noms,  
       count(distinct LOCALITE) as Localites,  
       count(distinct CAT)   as Categories  
  from CLIENT;
```

Numeros	Noms	Localites	Categories
16	15	7	4

# Les fonctions agrégatives

*Attention aux ensembles vides*

```
select count(*) as Nombre, sum(COMPTE) as Somme,  
       max(CAT)  as Max  
from   CLIENT  
where  LOCALITE = 'Alger';
```

Nombre	Nombre	Max
0	<null>	<null>

# Résumé

Fonction	Objectif
AVG ( [DISTINCT] <i>expr</i> )	Moyenne de <i>expr</i> (nombre).
COUNT ( { *   [DISTINCT ] <i>expr</i> } )	Nombre de lignes (* toutes les lignes, <i>expr</i> pour les colonnes non nulles).
GROUP_CONCAT ( <i>expr</i> )	Composition d'un ensemble de valeurs.
MAX ( [DISTINCT] <i>expr</i> )	Maximum de <i>expr</i> (nombre, date, chaîne).
MIN ( [DISTINCT] <i>expr</i> )	Minimum de <i>expr</i> (nombre, date, chaîne).
STDDEV ( <i>expr</i> )	Écart type de <i>expr</i> (nombre).
SUM ( [DISTINCT] <i>expr</i> )	Somme de <i>expr</i> (nombre).
VARIANCE ( <i>expr</i> )	Variance de <i>expr</i> (nombre).

# Insertion multiple

## Création et insertion

```
CREATE TABLE  
NomsetHVoldesPilotes  
(nom VARCHAR(16),  
nbHVol DECIMAL(7,2),  
compa CHAR(4));
```

```
INSERT INTO NomsetHVoldesPilotes  
    SELECT nom,nbHVol,compa  
        FROM Pilote;
```

## Requête et résultat

```
mysql> SELECT * FROM NomsetHVoldesPilotes;
```

nom	nbHVol	compa
Gratien Viel	450.00	AF
Didier Donsez	0.00	AF
Richard Grin	1000.00	SING
Placide Fresnais	2450.00	CAST
Daniel Vielle	NULL	AF

# Principe d'une jointure

Animal

<i>id</i>	<i>sexe</i>	<i>nom</i>	<i>race_id</i>	<i>espece_id</i>
24	M	Cartouche	NULL	1
25	M	Zambo	1	1
33	M	Caribou	4	2

Especie

<i>id</i>	<i>nom_courant</i>	<i>nom_latin</i>
1	Chien	Canis canis
2	Chat	Felix silvestris

Jointure Animal-Especie

<i>id</i>	<i>sexe</i>	<i>nom</i>	<i>race_id</i>	<i>espece_id</i>	<i>id</i>	<i>nom_courant</i>	<i>nom_latin</i>
24	M	Cartouche	NULL	1	1	Chien	Canis canis
25	M	Zambo	1	1	1	Chien	Canis canis
33	M	Caribou	4	2	2	Chat	Felix silvestris

# Classification

- Une jointure peut s'écrire, dans une requête SQL, de différentes manières :
  - « relationnelle » (aussi appelée « SQL89 » pour rappeler la version de la norme SQL)
  - « SQL2 » (aussi appelée « SQL92 »)
  - « procédurale » (qui qualifie la structure de la requête)
  - « mixte » (combinaison des trois approches précédentes)

# Types de jointure

- **Les jointures internes (*inner joins*) :**
  - L'équijointure (*equi join*) est la plus connue, elle utilise l'opérateur d'égalité dans la clause de jointure.
  - L'autojointure (*self join*) est un cas particulier de l'équijointure, qui met en oeuvre deux fois la même table (des alias de tables permettront de distinguer les enregistrements entre eux).
- **La jointure externe (*outer join*)**, la plus compliquée, qui favorise une table (dite « dominante ») par rapport à l'autre (dite « subordonnée »). Les lignes de la table dominante sont retournées même si elles ne satisfont pas aux conditions de jointure.

# Les jointures procédurales

# Jointures procédurales

---

Les jointures procédurales sont écrites par des requêtes qui contiennent des sous-interrogations (SELECT imbriqué). Chaque clause FROM ne contient qu'une seule table.

```
SELECT colonnesTable1 FROM [nomBase.]nomTable1
    WHERE colonne(s) | expression(s) { IN | = | opérateur }
        (SELECT colonne(s) delaTable2 FROM [nomBase.]nomTable2
            WHERE colonne(s) | expression(s) { IN | = | opérateur }
                (SELECT ...)
                [AND (conditionsTable2)]
            )
        [AND (conditionsTable1)];
```

---

# Les sous-requêtes

*Les numéros des clients de Namur :*

```
select NCLI  
from   CLIENT  
where  LOCALITE = 'Namur' ;
```

NCLI
B062
C123
L422
S127

# Les sous-requêtes

*Les numéros des clients de Namur :*

NCLI
B062
C123
L422
S127

*Les numéros des commandes des clients de Namur :*

```
select NCOM, DATECOM  
from   COMMANDE  
where  NCLI in ('C123','S127','B062','L422');
```

ne marche  
qu'une fois

*mieux :*

```
select NCOM, DATECOM  
from   COMMANDE  
where  NCLI in (select NCLI  
                  from   CLIENT  
                  where  LOCALITE = 'Namur');
```

marchera  
toujours

```
select *
from PRODUIT
where NPRO in
    (select NPRO
     from DETAIL
     where NCOM in
         (select NCOM
          from COMMANDE
          where NCLI in
              (select NCLI
               from CLIENT
               where LOCALITE='Namur')));
```

# Les sous-requêtes

```
select *  
from PRODUIT  
where NPRO in  
  (select NPRO  
   from DETAIL  
   where NCOM in  
     (select NCOM  
      from COMMANDE  
      where NCLI in  
        (select NCLI  
         from CLIENT  
         where LOCALITE='Namur')));
```

→ les clients de Namur

→ les commandes des clients de Namur

→ les détails des commandes des clients de Namur

→ les produits référencés par les détails des commandes des clients de Namur

# Condition d'association

Une condition *in (sous-requête)* correspond le plus souvent à une condition d'association = *qui sont associés à ...*

```
select *
from   T
where  CT in (select CS
                from   S
                where  <condition>);
```

"on recherche les *T* qui sont associés à des *S* qui ..."

# Sous-interrogations synchronisées

```
select NCLI, NOM, LOCALITE, COMPTE  
from CLIENT as C  
where COMPTE > (select avg(COMPTE)  
                  from CLIENT  
                  where LOCALITE = C.LOCALITE) ;
```

"C" est  
un alias de table

C est une variable qui à chaque instant référence la ligne courante de CLIENT dans la requête externe (on suppose qu'on examine successivement les lignes de CLIENT)

# Références multiples

Condition d'association quantifiée :  
*recherche des commandes d'au moins 3 détails*

```
select NCOM, DATECOM, NCLI  
from   COMMANDE C  
where  (select count(*)  
        from DETAIL  
        where NCOM = C.NCOM) >= 3;
```

# Les quantificateurs ensemblistes

## exists et not exists

le prédictat **exists(E)**, où E est une sous-requête,  
est vrai si l'ensemble désigné par E est *non vide*

*quels sont les produits pour lesquels il existe au moins un détail ?*

```
select NPRO, LIBELLE  
from PRODUIT as P  
where exists (select *  
              from DETAIL  
              where NPRO = P.NPRO) ;
```

le prédictat **not exists(E)**,  
est vrai si l'ensemble désigné par E est *vide*

# Les quantificateurs ensemblistes

## all et any

*quelles sont les commandes **qui spécifient** la plus petite quantité de PA60 ?*

```
select distinct NCOM
from   DETAIL
where  QCOM <= all (select QCOM
                     from   DETAIL
                     where  NPRO = 'PA60')
and    NPRO = 'PA60' ;
```

```
select distinct NCOM
from   DETAIL
where  QCOM = (select min(QCOM)
                  from   DETAIL
                  where  NPRO = 'PA60')
and    NPRO = 'PA60' ;
```

variante

# Les quantificateurs ensemblistes

*quels sont les commandes qui ne spécifient pas la plus petite quantité de PA60 ?*

```
select *
from   DETAIL
where  QCOM > any (select QCOM
                     from   DETAIL
                     where  NPRO = 'PA60')
and    NPRO = 'PA60' ;
```

```
select distinct NCOM
from   DETAIL
where  QCOM > (select min(QCOM)
                  from   DETAIL
                  where  NPRO = 'PA60')
and    NPRO = 'PA60' ;
```

variante

# Les quantificateurs ensemblistes

```
select *  
from CLIENT  
where NCLI in (select NCLI  
                from COMMANDE  
                where DATECOM = '12-09-2009') ;
```

plus concise

```
select *  
from CLIENT C  
where exists (select *  
              from COMMANDE M  
              where M.NCLI = C.NCLI  
              and DATECOM = '12-09-2009') ;
```

plus générale

# Les jointures relationnelles

# Les jointures

La **jointure** permet de produire une table constituée de données extraites de plusieurs tables :

The diagram illustrates a left outer join between two tables: 'table COMMANDE' and 'table CLIENT'. The 'table COMMANDE' is represented by a red-bordered grid of columns NCOM and DATECOM. The 'table CLIENT' is represented by a red-bordered grid of columns NCLI, NOM, and LOCALITE. An arrow labeled 'colonne commune' points from the NCLI column of the CLIENT table towards the NCLI column of the COMMANDE table, indicating that the join is based on the NCLI column. The resulting joined table contains all rows from the COMMANDE table and matching rows from the CLIENT table where NCLI values are common.

NCOM	DATECOM	NCLI	NOM	LOCALITE
30178	21/12/2008	K111	VANBIST	Lille
30179	22/12/2008	C400	FERARD	Poitiers
30182	23/12/2008	S127	VANDERKA	Namur
30184	23/12/2008	C400	FERARD	Poitiers
30185	2/01/2009	F011	PONCELET	Toulouse
30186	2/01/2009	C400	FERARD	Poitiers
30188	3/01/2009	B512	GILLET	Toulouse

```
select NCOM, DATECOM, CLIENT.NCLI, NOM, LOCALITE  
from   COMMANDE, CLIENT  
where  COMMANDE.NCLI = CLIENT.NCLI;
```

# Structure d'une requête de jointure

NCOM	DATECOM	NCLI	NOM	LOCALITE
30178	21/12/2008	K111	VANBIST	Lille
30179	22/12/2008	C400	FERARD	Poitiers
30182	23/12/2008	S127	VANDERKA	Namur
30184	23/12/2008	C400	FERARD	Poitiers
30185	2/01/2009	F011	PONCELET	Toulouse
30186	2/01/2009	C400	FERARD	Poitiers
30188	3/01/2009	B512	GILLET	Toulouse

```
select NCOM , DATECOM, CLIENT.NCLI, NOM, LOCALITE  
from COMMANDE, CLIENT  
where COMMANDE.NCLI = CLIENT.NCLI
```

préfixe nécessaire car ambiguïté

plusieurs tables

condition de jointure

# Autres exemples de jointure

```
select CLIENT.NCLI, NOM, DATECOM, NPRO  
from   CLIENT, COMMANDE, DETAIL  
where  CLIENT.NCLI = COMMANDE.NCLI  
and    COMMANDE.NCOM = DETAIL.NCOM;
```

jointure de 3 tables

```
select NCOM, CLIENT.NCLI, DATECOM, NOM, ADRESSE  
from   COMMANDE, CLIENT  
where  COMMANDE.NCLI = CLIENT.NCLI  
and    CAT = 'C1'  
and    DATECOM < '23-12-2009' ;
```

condition de jointure  
+  
conditions de sélection

# Produit relationnel

```
select NCOM, CLIENT.NCLI, DATECOM, NOM, ADRESSE  
from   COMMANDE, CLIENT;
```

pas de condition  
de jointure !

## Produit relationnel :

chaque ligne de COMMANDE est couplée avec chaque ligne de CLIENT

*requête valide mais d'utilité réduite*

# Sous-requête ou jointure ?

'eut-on remplacer une sous-requête par une jointure ?

```
select NCOM,DATECOM  
from   COMMANDE  
where  NCLI in (select NCLI  
                  from   CLIENT  
                  where  LOCALITE = 'Poitiers');
```

=

```
select NCOM,DATECOM  
from   COMMANDE, CLIENT  
where  COMMANDE.NCLI = CLIENT.NCLI  
and    LOCALITE = 'Poitiers';
```

# Valeurs dérivées dans une jointure

```
select NCOM, D.NPRO, QCOM*PRI  
from DETAIL D, PRODUIT P  
where D.NPRO = P.NPRO;
```

```
select 'Montant commande 30184 = ', sum(QCOM*PRI  
from DETAIL D, PRODUIT P  
where D.NCOM = '30184'  
and D.NPRO = P.NPRO;
```

# Construction dynamique d'une table dans la clause FROM

Introduite dans SQL2, la possibilité de construire dynamiquement une table dans la clause FROM d'une requête est opérationnelle sous MySQL.

```
SELECT listeColonnes  
      FROM table1 aliasTable1, (SELECT... FROM table2 WHERE...) aliasTable2  
      [ WHERE (conditionsTable1etTable2) ];
```

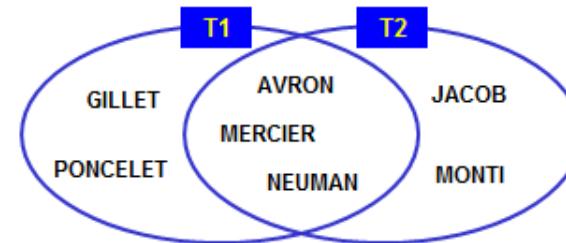
Requête et tables évaluées dans le FROM	Résultat										
<pre>SELECT a.compa "Comp", a.nbpil/b.total*100 "%Pilote"       FROM (SELECT compa, COUNT(*) nbpil               FROM Pilote GROUP BY compa) a,             (SELECT COUNT(*) total FROM Pilote) b;</pre> <p>a</p> <table border="1"><tr><th>compa</th><th>nbpil</th></tr><tr><td>AF</td><td>3</td></tr><tr><td>SING</td><td>1</td></tr><tr><td></td><td>1</td></tr></table> <p>b</p> <table border="1"><tr><th>total</th></tr><tr><td>5</td></tr></table>	compa	nbpil	AF	3	SING	1		1	total	5	+-----+-----+   Comp   %Pilote   +-----+-----+   NULL   20.0000     AF     60.0000     SING   20.0000   +-----+-----+
compa	nbpil										
AF	3										
SING	1										
	1										
total											
5											

# Opérations ensemblistes

Opérateurs **ensemblistes** entre 2 tables **sans doublons**

T1
NOM
GILLET
AVRON
MERCIER
PONCELET
NEUMAN

T2
NOM
MONTI
NEUMAN
JACOB
MERCIER
AVRON



```
select NOM  
from T1  
union  
select NOM  
from T2
```

```
select NOM  
from T1  
intersect  
select NOM  
from T2
```

```
select NOM  
from T1  
except  
select NOM  
from T2
```

NOM
GILLET
AVRON
MERCIER
PONCELET
NEUMAN
MONTI
JACOB

NOM
AVRON
MERCIER
NEUMAN

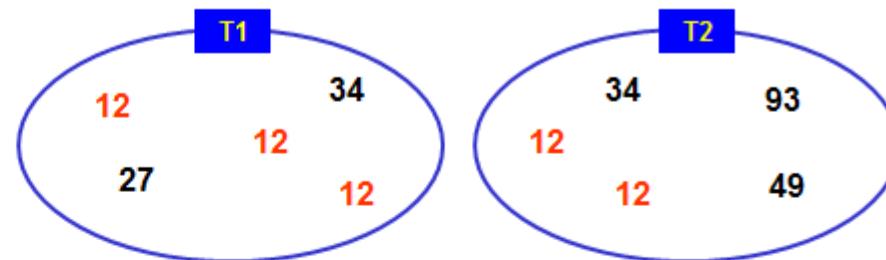
NOM
GILLET
PONCELET

# Opérateurs ensemblistes

Opérateurs **ensemblistes** entre 2 tables **avec doublons**

T1	NUM
	12
	34
	27
	12
	12

T2	NUM
	93
	12
	34
	12
	49



```
select NUM  
from T1  
union  
select NUM  
from T2
```

```
select NUM  
from T1  
intersect  
select NUM  
from T2
```

```
select NUM  
from T1  
except  
select NUM  
from T2
```

```
select NUM  
from T2  
except  
select NUM  
from T1
```

NUM
12
34
27
93
49

NUM
34
12

NUM
27

NUM
93
49

# L'intersection dans MySQL

AvionsdeAF

immat	typeAvion	nbHVol
F-WTSS	Concorde	6570
F-GLFS	A320	3500
F-GTMP	A340	

AvionsdeSING

immatriculation	typeAv	PrixAchat
S-ANSI	A320	104 500
S-AVEZ	A320	156 000
S-SMILE	A330	198 000
F-GTMP	A340	204 500

## Besoin

Quels sont les types d'avions que les deux compagnies exploitent en commun ?

## Requête

```
SELECT DISTINCT typeAvion FROM AvionsdeAF  
WHERE typeAvion IN  
(SELECT typeAv FROM AvionsdeSING);
```

```
+-----+  
| typeAvion |  
+-----+  
| A320      |  
| A340      |  
+-----+
```

# L'union dans MySQL

Avion de AF

immat	typeAvion	nbHVol
F-WTSS	Concorde	6570
F-GLFS	A320	3500
F-GTMP	A340	

Avion de SING

immatriculation	typeAv	PrixAchat
S-ANSI	A320	104 500
S-AVEZ	A320	156 000
S-SMILE	A330	198 000
F-GTMP	A340	204 500

## Besoin

Quels sont tous les types d'avions que les deux compagnies exploitent ?

## Requête

```
SELECT typeAvion FROM AvionsdeAF
UNION
SELECT typeAv FROM AvionsdeSING;
+-----+
| typeAvion |
+-----+
| A320      |
| A340      |
| Concorde  |
| A330      |
+-----+
```

# UNION ALL

AviondeAF

immat	typeAvion	nbHVol
F-WTSS	Concorde	6570
F-GLFS	A320	3500
F-GTMP	A340	

AviondeSING

immatriculation	typeAv	PrixAchat
S-ANSI	A320	104 500
S-AVEZ	A320	156 000
S-SMILE	A330	198 000
F-GTMP	A340	204 500

## Besoin

Même requête avec les duplicates. On extrait les types de la compagnie 'AF', suivis des types de la compagnie 'SING'.

## Requête

```
SELECT typeAvion FROM AvionsdeAF  
UNION ALL  
SELECT typeAv FROM AvionsdeSING;
```

typeAvion
A320
A340
Concorde
A340
A320
A320
A330

# La différence dans MySQL

AvionsdeAF

immat	typeAvion	nbHVol
F-WTSS	Concorde	6570
F-GLFS	A320	3500
F-GTMP	A340	

AvionsdeSING

immatriculation	typeAv	PrixAchat
S-ANSI	A320	104 500
S-AVEZ	A320	156 000
S-SMILE	A330	198 000
F-GTMP	A340	204 500

## Besoin

Quels sont les types d'avions exploités par la compagnie 'AF', mais pas par 'SING' ?

## Requête

```
SELECT DISTINCT typeAvion FROM AvionsdeAF
  WHERE typeAvion NOT IN
        (SELECT typeAv FROM AvionsdeSING);
+-----+
| typeAvion |
+-----+
| Concorde |
+-----+
```

Quels sont les types d'avions exploités par la compagnie 'SING,' mais pas par 'AF' ?

```
SELECT DISTINCT typeAv FROM AvionsdeSING
  WHERE typeAv NOT IN
        (SELECT typeAvion FROM AvionsdeAF);
+-----+
| typeAv |
+-----+
| A330   |
+-----+
```

# Ordonner les résultats

- Le résultat d'une requête contenant des opérateurs ensemblistes est trié, par défaut, par ordre croissant, sauf avec l'opérateur UNION ALL.

Technique	Requête
Nom de la colonne.	<pre>SELECT typeAvion FROM AvionsdeAF UNION SELECT typeAv FROM AvionsdeSING ORDER BY typeAvion DESC ;</pre>
Position de la colonne.	<pre>... ORDER BY 1 DESC ;</pre>
SELECT dans le FROM.	<pre>SELECT T.typeAvion FROM (SELECT typeAvion FROM AvionsdeAF UNION SELECT typeAv FROM AvionsdeSING) T ORDER BY T.typeAvion DESC;</pre>
	<pre>+-----+   typeAvion   +-----+   Concorde      A340          A330          A320        +-----+</pre>

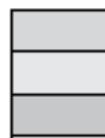
# La division

- La division est un opérateur algébrique et non ensembliste. Cet opérateur est semblable sur le principe à l'opération qu'on apprend au CM2. La division est un opérateur binaire comme la jointure, car il s'agit de diviser une table (ou partie de) par une autre table (ou partie de).

T1

Jospin	
Juppé	
Juppé	
Baudis	
Baudis	
Baudis	
Chirac	
Chirac	
Chirac	

T2



Quotient

Chirac

Quels sont les enregistrements de T1 qui sont associés à « tous les » enregistrements de T2 ?

*Réponse : Chirac*

Quotient = (T1 - Reste) / T2  
(ici Reste n'est pas nul)

# Exemple

- « Quels sont les avions affrétés par toutes les compagnies françaises ? ».

Affretements

immat	typeAv	compa	dateAff
A1	A320	SING	1965-05-13
A2	A340	AF	1968-06-22
A3	Mercure	AF	1965-02-05
A4	A330	ALIB	1965-01-16
A3	Mercure	ALIB	1942-03-05
A3	Mercure	SING	1987-03-01

Compagnie

comp	nomComp	pays
AF	Air France	F
ALIB	Air Lib	F
SING	Singapore AL	SG

Résultat

immat	typeAv
A3	Mercure

# Classification

- **Division inexacte** (le reste n'est pas nul) :
  - un ensemble est seulement inclus dans un autre ( $A \in B$ ). La question à programmer serait : « *Quels sont les avions affrétés par toutes les compagnies françaises ?* », sans préciser si les avions ne doivent pas être aussi affrétés par des compagnies étrangères. L'avion ('A3', 'Mercure') répondrait à cette question, que la dernière ligne de la table Affretements soit présente ou pas.
- **Division exacte** (le reste est nul) :
  - les deux ensembles sont égaux ( $B=A$ ). La question à programmer serait : « *Quels sont les avions affrétés exactement (ou uniquement) par toutes les compagnies françaises ?* ». L'avion ('A3', 'Mercure') répondrait à cette question si la dernière ligne de la table Affretements était inexistante. Les lignes concernées dans les deux tables sont grises.

# La division inexacte en SQL

Pour programmer le fait qu'un ensemble est seulement inclus dans un autre (ici  $A \subset B$ ), il faut qu'il n'existe pas d'élément dans l'ensemble  $\{A-B\}$ .

```
Parcours de tous les avions
SELECT DISTINCT immat, typeAv FROM Affretements aliasAff
WHERE NOT EXISTS
  (SELECT DISTINCT comp FROM Compagnie WHERE pays = 'F'
   AND comp NOT IN
    (SELECT compa FROM Affr etements WHERE immat = aliasAff.immat )) ;
          ↑ Ensemble A de référence
          ↑ Ensemble B à comparer
```

# La division exacte en SQL

- Pour programmer le fait qu'un ensemble est strictement égal à un autre (ici  $A=B$ ), il faut qu'il n'existe ni d'élément dans l'ensemble  $\{A-B\}$  ni dans l'ensemble  $\{B-A\}$ .

Parcours de tous les avions

```
SELECT DISTINCT immat, typeAv FROM Affrètements aliasAff
WHERE NOT EXISTS
    (SELECT comp FROM Compagnie WHERE pays = 'F' '
        AND comp NOT IN
            (SELECT compa FROM Affretements WHERE immat = aliasAff.immat ))
AND NOT EXISTS
    (SELECT compa FROM Affretelements WHERE immat = aliasAff.immat
        AND compa NOT IN
            (SELECT comp FROM Compagnie WHERE pays = 'F' ));
```

$A-B$

$B-A$

# Les données groupées

LOCALITES			
NCLI	NOM	LOCALITE	COMPTE
F400	JACOB	Bruxelles	0
B332	MONTI	Genève	0
K111	VANBIST	Lille	720
S127	VANDERKA	Namur	-4580
L422	FRANCK	Namur	0
C123	MERCIER	Namur	-2300
B062	GOFFIN	Namur	-3200
S712	GUILLAUME	Paris	0
F010	TOUSSAINT	Poitiers	0
B112	HANSENNE	Poitiers	1250
C400	FERARD	Poitiers	350
C003	AVRON	Toulouse	-1700
B512	GILLET	Toulouse	-8700
F011	PONCELET	Toulouse	0
K729	NEUMAN	Toulouse	0
D063	MERCIER	Toulouse	-2250

→ le groupe des clients de Genève

→ le groupe des clients de Namur

→ le groupe des clients de Poitiers

# Les données groupées

```
select LOCALITE,  
       count(*) as NOMBRE_CLIENTS,  
       avg(COMPTE) as MOYENNE_COMPTE  
  from CLIENT  
group by LOCALITE;
```

LOCALITE	NOMBRE_CLIENTS	MOYENNE_COMPTE
Bruxelles	1	0.00
Geneve	1	0.00
Lille	1	720.00
Namur	4	-2520.00
Paris	1	0.00
Poitiers	3	533.33
Toulouse	5	-2530.00

→ le groupe des clients de Genève

→ le groupe des clients de Namur

→ le groupe des clients de Poitiers

# Sélection de groupes

```
select LOCALITE, count(*) , avg(COMPTE)
from CLIENT
group by LOCALITE
having count(*) >= 3;
```

LOCALITE	count(*)	avg(COMPTE)
Namur	4	-2520.00
Poitiers	3	533.33
Toulouse	5	-2530.00

```
select NCLI, count(*)
from COMMANDE
group by NCLI
having count(*) >= 2;
```

# Autre exemple avec Group By

```
/* Data for the table employees */
INSERT INTO employees(name, salary, department_id) VALUES
('jack','3000.00', 1),
('mary','2500.00', 2),
('nichole','4000.00', 1),
('angie','5000.00', 2),
('jones','5000.00', 3);
```

```
/* Get number of employees for each department using GROUP BY */
SELECT department_id, COUNT(employee_id) AS employee_count
FROM employees
GROUP BY department_id;
```

department_id	employee_count
1	2
2	2
3	1

# Exemple avec Having

```
/* Data for the table employees */
INSERT INTO employees(name, salary, department_id) VALUES
('jack','3000.00', 1),
('mary','2500.00', 2),
('nichole','4000.00', 1),
('angie','5000.00', 2),
('jones','5000.00', 3);

/* Get number of employees for each department using GROUP BY &
 * the number of employees are greater than or equal to 2. */
SELECT department_id, COUNT(employee_id) AS employee_count
FROM employees
GROUP BY department_id
HAVING employee_count >= 2;
```

department_id	employee_count
1	2
2	2

# Technique de vue

```
CREATE VIEW v_HighSalaryEmployees AS  
  SELECT ename, salary FROM employees  
 WHERE salary > 4000;
```

```
CREATE VIEW v_LowSalaryEmployees AS  
  SELECT ename, salary FROM employees  
 WHERE salary < 3000;
```

```
mysql> SELECT * from v_HighSalaryEmployees;  
+-----+-----+  
| ename | salary |  
+-----+-----+  
| angie | 5000.00 |  
| jones | 5000.00 |  
+-----+-----+  
2 rows in set (0.00 sec)
```

# Autre syntaxe pour les jointures : Cross join

```
/* Cross Join Option #1 */  
SELECT 'Cross Join', e.ename, e.salary, d.dname  
FROM employees AS e, departments AS d;  
  
/* Cross Join Option #2 */  
SELECT 'Cross Join', e.ename, e.salary, d.dname  
FROM employees AS e CROSS JOIN departments AS d;
```

department_id	dname
1	Engineering
2	Sales
3	Marketing
4	HR

employee_id	ename	department_id	salary
1	jack	1	3000.00
2	mary	2	2500.00
3	nichole	1	4000.00
4	angie	2	5000.00
5	jones	3	5000.00
6	newperson	NULL	5000.00

Cross Join	ename	salary	dname
Cross Join	jack	3000.00	Engineering
Cross Join	jack	3000.00	Sales
Cross Join	jack	3000.00	Marketing
Cross Join	jack	3000.00	HR
Cross Join	mary	2500.00	Engineering
Cross Join	mary	2500.00	Sales
Cross Join	mary	2500.00	Marketing
Cross Join	mary	2500.00	HR
Cross Join	nichole	4000.00	Engineering
Cross Join	nichole	4000.00	Sales
Cross Join	nichole	4000.00	Marketing
Cross Join	nichole	4000.00	HR
Cross Join	angie	5000.00	Engineering
Cross Join	angie	5000.00	Sales
Cross Join	angie	5000.00	Marketing
Cross Join	angie	5000.00	HR
Cross Join	jones	5000.00	Engineering
Cross Join	jones	5000.00	Sales
Cross Join	jones	5000.00	Marketing
Cross Join	jones	5000.00	HR
Cross Join	newperson	5000.00	Engineering
Cross Join	newperson	5000.00	Sales
Cross Join	newperson	5000.00	Marketing
Cross Join	newperson	5000.00	HR

24 rows in set (0.00 sec)

# Inner join (jointure interne)

```
SELECT *                                -- comme d'habitude, vous  
sélectionnez les colonnes que vous voulez  
FROM nom_table1  
[INNER] JOIN nom_table2                -- INNER explicite le  
fait qu'il s'agit d'une jointure interne, mais c'est facultatif  
    ON colonne_table1 = colonne_table2   -- sur quelles colonnes  
se fait la jointure                      -- vous pouvez mettre  
colonne_table2 = colonne_table1, l'ordre n'a pas d'importance  
  
[WHERE ...]  
[ORDER BY ...]                        -- les clauses habituelles  
sont bien sûr utilisables !  
[LIMIT ...]
```

# Inner join

```
/* Inner Join Option #1 */
```

```
SELECT 'Inner Join', employees.ename, employees.salary, departments.dname  
FROM employees, departments  
WHERE employees.department_id=departments.department_id;
```

```
/* Inner Join Option #2 */
```

```
SELECT 'Inner Join', employees.ename, employees.salary, departments.dname  
FROM employees  
JOIN departments  
WHERE employees.department_id=departments.department_id;
```

```
/* Inner Join Option #3 */
```

```
SELECT 'Inner Join', employees.ename, employees.salary, departments.dname  
FROM employees  
INNER JOIN departments  
WHERE employees.department_id=departments.department_id;
```

```
/* Inner Join Option #4 */
```

```
SELECT 'Inner Join', employees.ename, employees.salary, departments.dname  
FROM employees  
INNER JOIN departments  
ON employees.department_id=departments.department_id;
```

# Résultat inner join

ename	salary	dname
jack	3000.00	Engineering
nichole	4000.00	Engineering
mary	2500.00	Sales
angie	5000.00	Sales
jones	5000.00	Marketing

5 rows in set (0.00 sec)

# Utilisation d'un alias

```
SELECT e.id,  
       e.description,  
       a.nom  
FROM Espece AS e          -- On donne l'alias "e" à Espece  
INNER JOIN Animal AS a    -- et l'alias "a" à Animal.  
      ON e.id = a.espece_id  
WHERE a.nom LIKE 'Ch%';
```

```
SELECT Espece.id AS id_espece,  
       Espece.description AS description_espece,  
       Animal.nom AS nom_bestiole  
FROM Espece  
INNER JOIN Animal  
      ON Espece.id = Animal.espece_id  
WHERE Animal.nom LIKE 'Ch%';
```

# Outer left join

```
/* Outer Join could be either LEFT JOIN or RIGHT JOIN */
```

```
/* Outer Join #1 - LEFT JOIN */
```

```
/* All records (actually fields of the records) of the "employees" table  
 * are included in the result set because the "employees" table is  
 * left side of the JOIN */
```

```
SELECT 'Outer Join - LEFT JOIN ', employees.ename, employees.salary,  
departments.dname  
FROM employees  
LEFT JOIN departments  
ON employees.department_id=departments.department_id;
```

# Résultat

// Notice that all records (actually fields of the records) of employees  
// table are included in the result set regardless of the match because  
// employees table is the left side of the outer left join.

Outer Join - LEFT JOIN	ename	salary	dname
Outer Join - LEFT JOIN	jack	3000.00	Engineering
Outer Join - LEFT JOIN	mary	2500.00	Sales
Outer Join - LEFT JOIN	nichole	4000.00	Engineering
Outer Join - LEFT JOIN	angie	5000.00	Sales
Outer Join - LEFT JOIN	jones	5000.00	Marketing
Outer Join - LEFT JOIN	newperson	5000.00	NULL

6 rows in set (0.00 sec)

# Outer right join

```
/* Outer Join could be either LEFT JOIN or RIGHT JOIN */  
  
/* Outer Join #2 - RIGHT JOIN */  
/* All records (actually fields of the records) of the "departments" table  
 * are included in the result set because the "departments" table is  
 * right side of the JOIN */  
SELECT 'Outer Join - RIGHT JOIN', employees.ename, employees.salary,  
departments.dname  
FROM employees  
RIGHT JOIN departments  
ON employees.department_id=departments.department_id;
```

# Résultat

// Notice that all records (actually fields of the records) of departments  
// table are included in the result set regardless of the match because  
// the departments table is the right side of the outer right join.

+-----+-----+-----+  Outer Join - RIGHT JOIN   ename   salary   dname	+-----+-----+-----+
Outer Join - RIGHT JOIN   jack   3000.00   Engineering	+-----+-----+-----+
Outer Join - RIGHT JOIN   nichole   4000.00   Engineering	+-----+-----+-----+
Outer Join - RIGHT JOIN   mary   2500.00   Sales	+-----+-----+-----+
Outer Join - RIGHT JOIN   angie   5000.00   Sales	+-----+-----+-----+
Outer Join - RIGHT JOIN   jones   5000.00   Marketing	+-----+-----+-----+
Outer Join - RIGHT JOIN   <b>NULL</b>   <b>NULL</b>   <b>HR</b>	+-----+-----+-----+

6 rows in set (0.00 sec)

# Jointures avec USING

- Lorsque les colonnes qui servent à joindre les deux tables ont **le même nom**, vous pouvez utiliser la clause **USING** au lieu de la clause **ON**.

```
SELECT *
FROM table1
[INNER | LEFT | RIGHT] JOIN table2 USING (colonneJ); -- colonneJ
est présente dans les deux tables
```

-- équivalent à

```
SELECT *
FROM table1
[INNER | LEFT | RIGHT] JOIN table2 ON Table1.colonneJ =
table2.colonneJ;
```

# Jointures naturelles

- Comme pour les jointures avec **USING**, il est possible d'utiliser les jointures naturelles dans le cas où les colonnes servant à la jointure ont **le même nom** dans les deux tables. Simplement, dans le cas d'une jointure naturelle, on ne donne pas la (les) colonne(s) sur laquelle (lesquelles) joindre les tables : c'est déterminé automatiquement.

```
SELECT *
FROM table1
NATURAL JOIN table2;
```

-- EST ÉQUIVALENT À

```
SELECT *
FROM table1
INNER JOIN table2
    ON table1.B = table2.B;
```

# Autre exemple

```
SELECT *
FROM table1
NATURAL JOIN table3;
```

-- EST ÉQUIVALENT À

```
SELECT *
FROM table1
INNER JOIN table3
ON table1.A = table3.A AND table1.C = table3.C;
```

# Jointures sans JOIN

```
SELECT *
FROM table1, table2
WHERE table1.colonne1 = table2.colonne2;
```

-- équivalent à

```
SELECT *
FROM table1
[INNER] JOIN table2
ON table1.colonne1 = table2.colonne2;
```

# Démo MySQL

# SELECT

Database: southwind

Table: products

productID INT	productCode CHAR(3)	name VARCHAR(30)	quantity INT	price DECIMAL(10,2)
1001	PEN	Pen Red	5000	1.23
1002	PEN	Pen Blue	8000	1.25
1003	PEN	Pen Black	2000	1.25
1004	PEC	Pencil 2B	10000	0.48
1005	PEC	Pencil 2H	8000	0.49

-- List all rows for the specified columns

```
mysql> SELECT name, price FROM products;
```

```
+-----+-----+
| name      | price   |
+-----+-----+
| Pen Red   | 1.23   |
| Pen Blue  | 1.25   |
| Pen Black | 1.25   |
| Pencil 2B | 0.48   |
| Pencil 2H | 0.49   |
+-----+-----+
```

5 rows in set (0.00 sec)

-- List all rows of ALL the columns. The wildcard \* denotes ALL columns

```
mysql> SELECT * FROM products;
```

```
+-----+-----+-----+-----+-----+
| productID | productCode | name      | quantity | price   |
+-----+-----+-----+-----+-----+
| 1001     | PEN        | Pen Red   | 5000    | 1.23   |
| 1002     | PEN        | Pen Blue  | 8000    | 1.25   |
| 1003     | PEN        | Pen Black | 2000    | 1.25   |
| 1004     | PEC        | Pencil 2B | 10000   | 0.48   |
| 1005     | PEC        | Pencil 2H | 8000    | 0.49   |
+-----+-----+-----+-----+-----+
```

5 rows in set (0.00 sec)

# Select sans table

```
mysql> SELECT 1+1;
+----+
| 1+1 |
+----+
|   2 |
+----+
1 row in set (0.00 sec)

mysql> SELECT NOW();
+-----+
| NOW()          |
+-----+
| 2012-10-24 22:13:29 |
+-----+
1 row in set (0.00 sec)

// Multiple columns
mysql> SELECT 1+1, NOW();
+----+-----+
| 1+1 | NOW()          |
+----+-----+
|   2 | 2012-10-24 22:16:34 |
+----+-----+
1 row in set (0.00 sec)
```

# Opérateurs de comparaison

```
mysql> SELECT name, price FROM products WHERE price < 1.0;
```

name	price
Pencil 2B	0.48
Pencil 2H	0.49

2 rows in set (0.00 sec)

```
mysql> SELECT name, quantity FROM products WHERE quantity <= 2000;
```

name	quantity
Pen Black	2000

1 row in set (0.00 sec)

# Les expressions régulières

```
-- "name" begins with 'PENCIL'  
mysql> SELECT name, price FROM products WHERE name LIKE 'PENCIL%';  
+-----+-----+  
| name      | price |  
+-----+-----+  
| Pencil 2B |  0.48 |  
| Pencil 2H |  0.49 |  
+-----+-----+  
  
-- "name" begins with 'P', followed by any two characters,  
-- followed by space, followed by zero or more characters  
mysql> SELECT name, price FROM products WHERE name LIKE 'P__ %';  
+-----+-----+  
| name      | price |  
+-----+-----+  
| Pen Red   |  1.23 |  
| Pen Blue  |  1.25 |  
| Pen Black |  1.25 |  
+-----+-----+
```

# Les opérateurs logiques

```
mysql> SELECT * FROM products WHERE quantity >= 5000 AND name LIKE 'Pen %';
+-----+-----+-----+-----+
| productID | productCode | name      | quantity | price |
+-----+-----+-----+-----+
|    1001 | PEN          | Pen Red   |     5000 |  1.23 |
|    1002 | PEN          | Pen Blue  |     8000 |  1.25 |
+-----+-----+-----+-----+

mysql> SELECT * FROM products WHERE quantity >= 5000 AND price < 1.24 AND name LIKE 'Pen %';
+-----+-----+-----+-----+
| productID | productCode | name      | quantity | price |
+-----+-----+-----+-----+
|    1001 | PEN          | Pen Red   |     5000 |  1.23 |
+-----+-----+-----+-----+

mysql> SELECT * FROM products WHERE NOT (quantity >= 5000 AND name LIKE 'Pen %');
+-----+-----+-----+-----+
| productID | productCode | name      | quantity | price |
+-----+-----+-----+-----+
|    1003 | PEN          | Pen Black |     2000 |  1.25 |
|    1004 | PEC          | Pencil 2B |    10000 |  0.48 |
|    1005 | PEC          | Pencil 2H |     8000 |  0.49 |
+-----+-----+-----+-----+
```

# IN, NOT IN

```
mysql> SELECT * FROM products WHERE name IN ('Pen Red', 'Pen Black');
```

productID	productCode	name	quantity	price
1001	PEN	Pen Red	5000	1.23
1003	PEN	Pen Black	2000	1.25

# BETWEEN, NOT BETWEEN

```
mysql> SELECT * FROM products  
      WHERE (price BETWEEN 1.0 AND 2.0) AND (quantity BETWEEN 1000 AND 2000);  
+-----+-----+-----+-----+  
| productID | productCode | name       | quantity | price   |  
+-----+-----+-----+-----+  
|     1003  |    PEN      | Pen Black |     2000  |  1.25  |  
+-----+-----+-----+-----+
```

# IS NULL, IS NOT NULL

```
mysql> SELECT * FROM products WHERE productCode IS NULL;  
Empty set (0.00 sec)
```

# ORDER BY

```
-- Order the results by price in descending order
```

```
mysql> SELECT * FROM products WHERE name LIKE 'Pen %' ORDER BY price DESC;
```

productID	productCode	name	quantity	price
1002	PEN	Pen Blue	8000	1.25
1003	PEN	Pen Black	2000	1.25
1001	PEN	Pen Red	5000	1.23

```
-- Order by price in descending order, followed by quantity in ascending (default) order
```

```
mysql> SELECT * FROM products WHERE name LIKE 'Pen %' ORDER BY price DESC, quantity;
```

productID	productCode	name	quantity	price
1003	PEN	Pen Black	2000	1.25
1002	PEN	Pen Blue	8000	1.25
1001	PEN	Pen Red	5000	1.23

# LIMIT

-- Display the first two rows

```
mysql> SELECT * FROM products ORDER BY price LIMIT 2;
```

productID	productCode	name	quantity	price
1004	PEC	Pencil 2B	10000	0.48
1005	PEC	Pencil 2H	8000	0.49

-- Skip the first two rows and display the next 1 row

```
mysql> SELECT * FROM products ORDER BY price LIMIT 2, 1;
```

productID	productCode	name	quantity	price
1001	PEN	Pen Red	5000	1.23

# AS

```
mysql> SELECT productID AS ID, productCode AS Code,
      name AS Description, price AS `Unit Price` -- Define aliases to be used as display names
      FROM products
      ORDER BY ID; -- Use alias ID as reference
+----+----+-----+-----+
| ID | Code | Description | Unit Price |
+----+----+-----+-----+
| 1001 | PEN | Pen Red | 1.23 |
| 1002 | PEN | Pen Blue | 1.25 |
| 1003 | PEN | Pen Black | 1.25 |
| 1004 | PEC | Pencil 2B | 0.48 |
| 1005 | PEC | Pencil 2H | 0.49 |
+----+----+-----+-----+
```

# CONCAT()

```
mysql> SELECT CONCAT(productCode, ' - ', name) AS `Product Description`, price FROM products;
+-----+-----+
| Product Description | price |
+-----+-----+
| PEN - Pen Red      | 1.23 |
| PEN - Pen Blue     | 1.25 |
| PEN - Pen Black    | 1.25 |
| PEC - Pencil 2B    | 0.48 |
| PEC - Pencil 2H    | 0.49 |
+-----+-----+
```

# DISTINCT

-- Without DISTINCT

```
mysql> SELECT price FROM products;
+-----+
| price |
+-----+
| 1.23 |
| 1.25 |
| 1.25 |
| 0.48 |
| 0.49 |
+-----+
```

-- With DISTINCT on price

```
mysql> SELECT DISTINCT price AS `Distinct Price` FROM products;
+-----+
| Distinct Price |
+-----+
|      1.23 |
|      1.25 |
|      0.48 |
|      0.49 |
+-----+
```

-- DISTINCT combination of price and name

```
mysql> SELECT DISTINCT price, name FROM products;
+-----+-----+
| price | name   |
+-----+-----+
| 1.23 | Pen Red |
| 1.25 | Pen Blue |
| 1.25 | Pen Black |
| 0.48 | Pencil 2B |
| 0.49 | Pencil 2H |
+-----+-----+
```

# Group by

```
mysql> SELECT * FROM products ORDER BY productCode, productID;
```

productID	productCode	name	quantity	price
1004	PEC	Pencil 2B	10000	0.48
1005	PEC	Pencil 2H	8000	0.49
1001	PEN	Pen Red	5000	1.23
1002	PEN	Pen Blue	8000	1.25
1003	PEN	Pen Black	2000	1.25

```
mysql> SELECT * FROM products GROUP BY productCode;
```

-- Only first record in each group is shown

productID	productCode	name	quantity	price
1004	PEC	Pencil 2B	10000	0.48
1001	PEN	Pen Red	5000	1.23

# Group by et fonctions d'aggrégation

```
-- Function COUNT(*) returns the number of rows selected
mysql> SELECT COUNT(*) AS `Count` FROM products;
      -- All rows without GROUP BY clause
+-----+
| Count |
+-----+
|     5 |
+-----+

mysql> SELECT productCode, COUNT(*) FROM products GROUP BY productCode;
+-----+-----+
| productCode | COUNT(*) |
+-----+-----+
| PEC         |      2 |
| PEN         |      3 |
+-----+-----+

-- Order by COUNT - need to define an alias to be used as reference
mysql> SELECT productCode, COUNT(*) AS count
      FROM products
      GROUP BY productCode
      ORDER BY count DESC;
+-----+-----+
| productCode | count |
+-----+-----+
| PEN         |     3 |
| PEC         |     2 |
+-----+-----+
```

# Having

```
mysql> SELECT
    productCode AS `Product Code`,
    COUNT(*) AS `Count`,
    CAST(AVG(price) AS DECIMAL(7,2)) AS `Average`
  FROM products
 GROUP BY productCode
 HAVING Count >=3;
 -- CANNOT use WHERE count >= 3
 +-----+-----+-----+
 | Product Code | Count | Average |
 +-----+-----+-----+
 | PEN          |     3 |      1.24 |
 +-----+-----+-----+
```

# Select with join

```
-- ANSI style: JOIN ... ON ...
mysql> SELECT products.name, price, suppliers.name
      FROM products
            JOIN suppliers ON products.supplierID = suppliers.supplierID
      WHERE price < 0.6;
+-----+-----+
| name | price | name      |
+-----+-----+
| Pencil 3B | 0.52 | ABC Traders |
| Pencil 6B | 0.47 | XYZ Company |
+-----+
-- Need to use products.name and suppliers.name to differentiate the two "names"

-- Join via WHERE clause (lagacy and not recommended)
mysql> SELECT products.name, price, suppliers.name
      FROM products, suppliers
      WHERE products.supplierID = suppliers.supplierID
            AND price < 0.6;
+-----+-----+
| name | price | name      |
+-----+-----+
| Pencil 3B | 0.52 | ABC Traders |
| Pencil 6B | 0.47 | XYZ Company |
+-----+
```

# Utilisation d'alias

```
-- Use aliases for column names for display
mysql> SELECT products.name AS `Product Name`, price, suppliers.name AS `Supplier Name`
   FROM products
      JOIN suppliers ON products.supplierID = suppliers.supplierID
     WHERE price < 0.6;
+-----+-----+
| Product Name | price | Supplier Name |
+-----+-----+
| Pencil 3B    |  0.52 | ABC Traders   |
| Pencil 6B    |  0.47 | XYZ Company   |
+-----+-----+

-- Use aliases for table names too
mysql> SELECT p.name AS `Product Name`, p.price, s.name AS `Supplier Name`
   FROM products AS p
      JOIN suppliers AS s ON p.supplierID = s.supplierID
     WHERE p.price < 0.6;
```

# Exemple avec trois tables

```
mysql> SELECT products.name AS `Product Name`, price, suppliers.name AS `Supplier Name`  
      FROM products_suppliers  
        JOIN products ON products_suppliers.productID = products.productID  
        JOIN suppliers ON products_suppliers.supplierID = suppliers.supplierID  
      WHERE price < 0.6;  
+-----+-----+-----+  
| Product Name | price | Supplier Name |  
+-----+-----+-----+  
| Pencil 3B     |  0.52 | ABC Traders   |  
| Pencil 3B     |  0.52 | QQ Corp       |  
| Pencil 6B     |  0.47 | XYZ Company   |  
+-----+-----+-----+  
  
-- Define aliases for tablenames too  
mysql> SELECT p.name AS `Product Name`, s.name AS `Supplier Name`  
      FROM products_suppliers AS ps  
        JOIN products AS p ON ps.productID = p.productID  
        JOIN suppliers AS s ON ps.supplierID = s.supplierID  
      WHERE p.name = 'Pencil 3B';  
+-----+-----+  
| Product Name | Supplier Name |  
+-----+-----+  
| Pencil 3B     | ABC Traders   |  
| Pencil 3B     | QQ Corp       |  
+-----+-----+
```

-- Using WHERE clause to join (legacy and not recommended)

```
mysql> SELECT p.name AS `Product Name`, s.name AS `Supplier Name`  
      FROM products AS p, products_suppliers AS ps, suppliers AS s  
     WHERE p.productID = ps.productID  
       AND ps.supplierID = s.supplierID  
       AND s.name = 'ABC Traders';
```

Product Name	Supplier Name
Pencil 3B	ABC Traders
Pencil 4B	ABC Traders
Pencil 5B	ABC Traders

