

SQL : Structured Query Language

Développé initialement par IBM, créé en 1974 et normalisé en 1987

Théoriquement, toutes les SGDB suivent la norme SQL. C'est le cas de MySQL, PostgreSQL, ...

Le SQL est un langage normalisé servant à exploiter des bases de données relationnelles (stockage de l'information/données décomposées et organisées dans des tables).

On distingue 4 groupes de commandes :

- DDL data definition language

CREATE, ALTER, DROP, RENAME

- DQL data query language

SELECT

- DML data manipulation language

INSERT, DELETE, UPDATE

- DCL data control language

GRANT, REVOKE, COMMIT, ROLLBACK

MySQL

Un serveur MySQL contient une ou plusieurs base de données, chaque base contient des/une table(s), structurée en colonne (ou champ) définis à sa création, et contient de 0 à N enregistrement(s) (lignes).

MySQL est un SGBD (système de gestion de base de données) il est distribué sous une double licence GPL et propriétaire.

MySQL est un serveur de base de données relationnelles SQL, orienté service de données, lecture, moins optimisé pour les opérations de mise à jour fréquentes fortement sécurisées. Il est multi-thread et multi-utilisateur

MySQL fait partie du quatuor LAMP, et ses variantes WAMP et MAMP. Wikipédia a utilisé MySQL jusqu'en 2012, il utilise maintenant un fork communautaire MariaDB. (les deux noms MySQL et MariaDB viennent des noms des filles de Monty Widenius cofondateur de MySQL, My et Maria).

MySQL AB a été acheté le 16 janvier 2008 par Sun Microsystems pour un milliard de dollars américains. En 2009, Sun Microsystems a été acquis par Oracle Corporation, mettant entre les mains d'une même société les deux produits concurrents que sont Oracle Database et MySQL. Ce rachat a été autorisé par la Commission européenne le 21 janvier 2010.

MySQL

Configuration pour l'utilisation de MySQL en ligne de commande :

Windows : il faut modifier la variable d'environnement (variables dynamiques utilisées par le système). On fait un clic droit sur le bureau, propriétés système, avancé et enfin « variables d'environnement »

Il faut ajouter aux variables d'environnement de Windows le chemin vers l'exécutable mysql (il se trouve dans le dossier bin de wamp) attention on ne doit mettre que le chemin du dossier de l'exécutable. Pour séparer les différents exécutables on utilise le « ; » dans le path de windows.

Mac : on peut dans un terminal taper la ligne suivante :

```
$ /Applications/MAMP/Library/bin/mysql -u root -p
```

On peut également créer un raccourci vers l'exécutable mysql, dans ce cas sous UNIX, on doit mettre un lien symbolique dans le dossier /usr/local/bin vers l'exécutable mysql.

Pour faire cela il faut être « root » on le fera avec la commande sudo

Attention, pour pouvoir utiliser sudo, il faut avoir un mot de passe pour le compte admin (menu Pomme > Préférence Système et cliquer sur Comptes). Voir [ici](#)

Et enfin on crée le lien symbolique :

```
user $ sudo ln -s /Applications/MAMP/Library/bin/mysql /usr/local/bin/mysql
```

Linux: il n'y a rien à faire, lorsqu'on installe LAMP on peut directement se connecter au serveur MySQL en ligne de commande.

On a un fichier de configuration global de MySQL dans WAMP : my.ini

On peut par exemple personnaliser le prompt lors de la connexion au serveur sous la section [mysql] de ce fichier on peut écrire :

```
[mysql]  
prompt=(\u@\u) [\d]\_mysql>\_
```

Par ailleurs, MySQL garde chaque base sous forme de dossiers et chaque tables sous forme de fichiers, voir dans le dossier data, le fichier db.opt précise l'encodage des caractères et la collation.

Organisation pour les exemples du cours (dans le dossier wamp/www si on veut faire du PHP avec MySQL) :

sql/

intro/

example.sql

Se connecter au serveur de base de données :

\$mysql -u root -p

mot de passe :

mysql>

Information sur le serveur de base de données

mysql> **ls**

Pour avoir un prompt indiquant sur quel serveur de base de données et avec quel utilisateur on est connecté

mysql> **\R \U [\d]>**

Si on utilise la base « shop » avec le compte root sur le serveur en localhost on aura :

root@localhost [shop]>

On peut configurer cela de manière permanente dans le fichier my.cnf, sous linux /etc/mysql/my.cnf et sinon regarder dans le dossier bin pour les autres systèmes...

prompt = \U [\d]_

De manière générale si on tape « ? » dans la console pour un service donné, cette instruction retourne l'ensemble de la documentation des commandes du programme.

Pour afficher toutes les bases de données du serveur MySQL

>SHOW DATABASES ;

Pour afficher toutes les tables d'une base de données

>USE mabase ;

>SHOW TABLES ;

jeu de caractères et collation

« Jeu de caractères » est un ensemble de lettres, signes de ponctuations et autres symboles auxquels on associe un numéro de code.

Unicode a pour objectif de fournir un jeu de caractères unique pour représenter toutes les langues. L'UTF8 est un codage de caractères conçu pour coder l'ensemble des caractères internationaux d'Unicode ; il reste compatible avec la norme ASCII. L'American Standard Code for Information Interchange (Code américain normalisé pour l'échange d'information).

L'UTF8 utilise un maximum de 4 octets par caractère. Par exemple, le « é » en UTF8 est codé sur deux octets : 11000011 10101001.

De par sa nature, UTF-8 est d'un usage de plus en plus courant sur l'Internet, et dans les systèmes devant échanger de l'information. Il s'agit également du standard Unicode le plus utilisé dans les écosystèmes GNU, Linux et compatibles, **donc très utilisé pour les sites web hébergés dans l'écosystème LAMP la plupart du temps.**

Collation (rassemblement et comparaison)

En plus du « jeu de caractères » MySQL permet de choisir comment seront classées (ORDER BY) les données : c'est ce que l'on appelle la collation (COLLATE).

Comment ordonner A,a, B,b , ordre des accents, ... ?

Supposons que nous ayons un jeu de caractères de 4 lettres : A,B,a,b avec l'ordre suivant A=0, B=1, a=2, c=3, si maintenant on veut comparer les lettres A et B il suffit de comparer le nombre 0 et 1 et donc A<B. Ce que nous faisons ici correspond à une collation binaire. Cependant, le problème devient un peu plus complexe si on doit comparer A et a, il faut si la collation est sensible à la casse définir d'autres règles (regrouper a,b et A et B). Dans la réalité, une collation possède de nombreuses règles pour comparer les lettres.

Dans notre cas on utilisera la collation utf8_general_ci, elle est insensible à la casse, ce qui permet de faire des recherches dans les tables plus rapidement.

Attention, MySQL ne supporte pas (encore) les caractères sur 4 octets comme 
mysql>INSERT INTO Foo (title) VALUES ('');

On peut voir l'encodage des caractères d'une colonne à l'aide de la commande hex() pour hexadécimal :
SELECT HEX(title) FROM Foo;

Introduction aux type SQL

MySQL est un langage typé, une colonne dans une table possède un type.

Type numérique : deux groupes types entiers et types décimaux.

Entiers :

foo INT

sur 4 octets (1 octet=8 bits par exemple 10100011), nombre compris entre -2147483648 et 2147483648

Attention, si la valeur que l'on enregistre dépasse le max ou le min de cet intervalle, alors MySQL met son max ou son min par défaut :

INSERT INTO Foo (foo) VALUES (9999999999); => enregistre en réalité 2147483648

foo BIGINT

8 octets => -9223372036854775808 et 9223372036854775808

foo TINYINT

1 octet => -127 à 127

On peut demander à MySQL de ne prendre que la plage des nombres positifs :

foo TINYINT UNSIGNED

1 octet => 0 à 255

Attention, lorsqu'on écrira INT(2), la valeur 2, par exemple, précise le nombre de chiffres à afficher, mais MySQL continue à stocker des nombres sur 4 octets. INT(11) correspond au nombre de chiffres du max ou du min de l'intervalle pour INT.

Décimaux

DECIMAL(n,v) pour stocker des nombres sur n chiffres significatifs ayant v chiffres après la virgule:

CREATE TABLE Foo (bar DECIMAL(5,3) NOT NULL) ;

Dans la table Foo on pourra stocker des valeurs dans la colonne bar ayant 5 chiffres en tout dont 3 après la virgule : 20.567 et non pas 200.890, par exemple. Attention, aux valeurs approchées par MySQL.

On a également les autres types suivants : NUMERIC, FLOAT

CHAR(x) et VARCHAR(x)

Pour stocker un texte court $x < 255$ octets on utilise CHAR et VARCHAR

CHAR(x)

Stocke (réserve) x octets quelle que soit la valeur que l'on place dans la colonne ayant ce type.

Attention, donc MySQL réserve un nombre d'octets fixe pour chaque champ.

On utilisera ce type pour une chaîne ayant **toujours la même longueur** par exemple une référence client de taille fixe, un code de marque...

VARCHAR(x) stocke jusqu'à x octets (0 à x octets) selon la taille du mot à stocker. Cette option alloue donc dynamiquement le nombre d'octets requis $\leq x$.

Remarque, si le texte est plus long prend plus de x octets \Rightarrow MySQL tronque le texte.

Pour les textes supérieurs à 255 octets on utilise le type

TEXT sur 2^{16} octets

BLOB est un type binaire capable de stocker des objets binaires de grande taille. Ici les chaînes de caractères seront sensibles à la casse...Nous ne l'aborderons pas dans le cours.

ENUM

Définit un ensemble de caractères autorisés de type chaîne de caractères :

title ENUM('foo', 'bar', 'baz')

Si on essaye d'enregistrer une valeur non autorisée MySQL enregistre " chaîne vide ou NULL

SET

title SET('foo', 'bar', 'baz')

Est équivalent à ENUM sauf qu'ici on peut stocker des couples 'foo, bar' ou 'foo', ou 'foo, bar, baz'

DATE

Attention au format

AAAA-MM-JJ ~ AAMMJJ ~ AAAA/MM/JJ ~ AAAA%MM%JJ

INSERT INTO Foo (title, date) **VALUES** ('€', '001202'); => 2000-12-02

Le mieux c'est de d'utiliser le premier format AAAA-MM-JJ

DATETIME

AAAA-MM-JJ HH:MM:SS

On ne parlera pas du **TIMESTAMP** de MySQL (mal implémenté), sachez seulement qu'en informatique c'est pour une date donnée « t » le temps en secondes qui s'est écoulé depuis la naissance d'UNIX (1 janvier 1970, 0h0min0s).

DDL (data definition language)

CREATE, ALTER, DROP, RENAME

Pour créer des objets de type base ou table on utilise CREATE, pour modifier ou supprimer ces objets on utilise DROP, ALTER ou RENAME (attention RENAME ne fonctionne pas sur les bases de données).

Convention de nommage des tables et base de données, pas de caractères spéciaux et espace, préfixer le nom des bases de données : db_blog, wp_wordpress. Pour les tables on les écrit en minuscule sans « s » ou en commençant par une majuscule et sans « s ».

CREATE DATABASE IF NOT EXISTS

`db_bar`

DEFAULT CHARACTER – ici on précise le jeu de caractère pour les tables de cette base de données

SET utf8

COLLATE utf8_general_ci; – ici on précise la collation en utf8 insensible à la casse pour les tables de cette base de données

USE db_bar ;

CREATE TABLE `foo` (

`foo_id` INT(11) AUTO_INCREMENT,

`client` INT NOT NULL,

`ref` VARCHAR(30) NOT NULL,

PRIMARY KEY (`foo_id`)

) ENGINE=InnoDB AUTO_INCREMENT=1 ;

Remarque pour le chapitre introduction

Pour l'insertion de données dans les tables on utilisera une commande du type :

INSERT INTO foo (client, `ref`) VALUES (1, 'raspberryPi');

Exercice

Pour la suite du cours on créera la base et la table ci-dessus (faire un copier coller dans la console).

Engine

Nous parlerons ici des deux moteurs suivants, sachant que la plupart du temps on s'occupera du premier :

ENGINE=InnoDB ou MyISAM

MyISAM est plus rapide et a été amélioré dans ses nouvelles versions, on ne peut mettre que des clefs primaires.

InnoDB fournit un moteur de base de données transactionnelle (compatible ACID), la possibilité de créer des clefs étrangères (foreign key), En fait ce moteur est très utilisé dans les gros sites devant optimiser les performances des accès aux données. (InnoDB est sous licence GNU/GPL avec certaines restrictions pour des options du moteur sous licence commerciale voir MySQL Pro)

DEFAULT indique une valeur par défaut si on ne précise rien à l'insertion d'une ligne pour cette colonne

```
foo_id INT(11) NOT NULL DEFAULT 0
```

```
mysql>ALTER TABLE foo MODIFY foo_id INT(11) NOT NULL DEFAULT 0 ;
```

```
mysql>INSERT INTO foo (title) VALUES ('php5') ;
```

```
+-----+-----+-----+-----+
| foo_id | title | date_foo | datetime_foo |
+-----+-----+-----+-----+
|    0  | php5  | NULL     | NULL         |
+-----+-----+-----+-----+
```

NOT NULL si aucune valeur n'est renseignée pour le champ, une valeur par défaut est ajoutée : " ou 0, ... selon le type de la colonne.

NULL si aucune valeur n'est ajoutée dans la colonne, MySQL met le type NULL dans le champ. Cela est utile dans certains cas, par exemple note de classe, sexe des animaux à la naissance,...

UNIQUE est un type d'index, ici deux valeurs identiques dans une colonne ne peuvent être enregistrées.

```
mysql>ALTER TABLE foo add `bar_unique` INT(11) UNIQUE;
```

```
mysql>INSERT INTO foo (bar_unique) VALUES (1), (2), (3) ;
```

```
mysql>INSERT INTO foo (bar_unique) VALUES (1), (2), (3) ;
```

```
ERROR 1062 (23000): Duplicate entry '1' for key 'bar_unique'
```

```
ALTER TABLE foo ADD email VARCHAR(50) ; ← création d'un champ
```

```
ALTER TABLE foo ADD CONSTRAINT un_email UNIQUE(email) ; ← création d'un index unique avec contrainte de nom
```

```
ALTER TABLE foo DROP INDEX un_email ; ← suppression de l'index unique
```

```
ALTER TABLE foo DROP email; ← suppression d'un champ
```

AUTO_INCREMENT valeur qui s'incrémente automatiquement à l'insertion.

INDEX sur une colonne ou plusieurs (composite), il permet de parcourir plus rapidement les données d'une colonne (indexée), sans index c'est comme rechercher dans un dictionnaire où les mots ne sont pas rangés par ordre alphabétique. Par exemple, pour un tableau de 1000 enregistrements la recherche d'un élément à partir d'une colonne indexée est 100 plus rapide.

MySQL cependant crée une structure supplémentaire BTREE dans la base, ceci peut prendre de l'espace mémoire.

```
ALTER TABLE foo ADD INDEX ind_ref (ref) ; ( on va voir cela dans le cours)
```

PRIMARY KEY clef primaire, elle peut être composite (plusieurs nom de colonnes) <=> UNIQUE NOT NULL INDEX

FOREIGN KEY clef secondaire (éventuellement composite) elle se réfère à un INDEX ou clef primaire. (pour le moteur InnoDB uniquement).

RENAME pour renommer le nom d'une table. Attention ne marche pas avec les noms de base de données.
RENAME TABLE foo to Foo ;

DROP [TABLE | DATABASE | VIEW] supprime un objet SQL
DROP TABLE Foo ;
DROP VIEW Summary
DROP DATABASE db_bar ;

ALTER permet d'effectuer des modifications structurelles sur un objet. Par exemples pour ajouter une colonne à une table Foo après la colonne ref :

ALTER TABLE foo ADD price DECIMAL(5,2) NOT NULL AFTER foo_id ;

Modifier une colonne :

ALTER TABLE foo MODIFY price DECIMAL(10,2) NOT NULL AFTER foo_id ;

Supprimer une clef secondaire, un index :

ALTER TABLE foo DROP FOREIGN KEY fk_client_id_client ;
ALTER TABLE foo DROP INDEX ind_foo;

Retourne la définition de la table (très pratique)

SHOW CREATE TABLE foo;

Fournit des informations sur les colonnes de la table

DESCRIBE TABLE foo ;

clefs primaire et secondaire

Définition de la clef primaire

Elle peut être composite et est équivalente à **UNIQUE + NOT NULL + UNIQUE**

PRIMARY KEY <=> INDEX + NOT NULL + UNIQUE

On ajoute souvent à nos tables des identifiants uniques (clefs primaires) : `foo_id`

```
mysql>ALTER TABLE foo ADD PRIMARY KEY (foo_id) ;
```

Attention, les clefs primaires ne sont pas obligatoires dans une table, cependant on en créera une systématiquement pour chaque table...En l'absence d'une clef primaire sur une table on fait une erreur conceptuelle en terme de base de données relationnelle.

Pour les clefs étrangères on ne peut en créer qu'avec le moteur InnoDB, leur principal objectif : **vérification de l'intégrité des données**. C'est un garde fou pour ne pas insérer dans les tables des choses incohérentes avec le reste des données dans les tables.

On reviendra sur les clefs secondaires plus loin dans le cours/TP, avec des précisions.

DQL : (data query language)

Interrogation des données **SELECT**

On passe beaucoup de temps en SQL à faire des SELECT, MySQL est d'ailleurs bien programmé pour faire ce type de commande, lorsqu'on fait un select on fait de l'extraction de données.

Projection : on ne sélectionne ci-dessous que la colonne ref de la table Foo :

SELECT ref FROM foo ;

Restriction : on utilise un where pour n'extraire qu'un sous ensemble de données de la table :

SELECT * FROM WHERE foo_id=1 ;

Si on fait un **SELECT** sur plus d'une table il s'agit d'une jointure.

MySQL crée à chaque fois que l'on fait un **SELECT** (extraction de données) une pseudo-table, elle n'est pas gardée en mémoire.

mysql>SELECT 'Hello World !' ;

J'ai perdu ma montre

mysql>SELECT SYSDATE() 'Ma montre' ;

J'aime les mathématiques

mysql>SELECT POWER(2,14), COS(PI()), DEGREES(PI()), EXP(1) ;

POWER(2,14)	COS(PI())	DEGREES(PI())	EXP(1)
16384	-1	180	2.718281828459045

Voici ci-dessous la structure simplifiée d'une requête SQL de type SELECT, tout ce qui est en crochet est optionnel.

```
SELECT
  select_expression,...
  [FROM table_references
  [WHERE where_definition]
  [ORDER BY {nom_de_colonne } [ASC | DESC]]
  [LIMIT [offset,] lignes]
```

Exemple de ce que l'on pourra écrire comme requête SQL, il faudra respecter l'ordre des clauses ci-dessous si on les utilise ensemble.

SELECT ref, title	← projection
FROM foo	← nom de la table
WHERE foo_id>1 AND status='publish'	← restriction sous ensemble des données de la table
ORDER BY date_crea ASC	← ordre ASC ou DESC classer les résultats
LIMIT 0,10	← 0 : offset 10 le nombre d'éléments du tableau

Clause WHERE, restriction

On peut utiliser les opérateurs suivants : =, <, >, <=, >=, <> (**différent**)

Les opérateurs logiques **AND**, **OR**, **NOT**

Tester les concordances de chaînes : **LIKE** % n'importe quel caractères 1,N, et _ 1 caractère

IS NULL teste si un champ n'a pas été affecté d'une valeur (donc si il est NULL)

BETWEEN teste l'appartenance à un intervalle BETWEEN 1 AND 20

IN teste l'appartenance à un groupe de valeurs données.

Exemples

```
SELECT * FROM Table1 WHERE Table1.principal NOT IN (SELECT principal FROM table2)
```

```
SELECT title FROM article WHERE title LIKE '%a%';
```

```
SELECT title, article_id FROM article WHERE article_id BETWEEN 1 AND 2;
```

```
SELECT title, article_id FROM article WHERE article_id IN (1,2,3);
```

Exercices (DQL)

Interrogation des données SELECT

1 Exercice DDL

Organisation

sql/

chap1/

compagny.sql

Dans un fichier écrire les définitions de la base de données « db_compagny » et de la table « Pilot », on donne ci-dessous la table qu'il faut créer. Cette table aura une clef primaire en plus pilot_id et le champ certificate sera de type unique.

On écrira le code SQL dans le fichier compagny.sql, on pourra utiliser la commande SOURCE de MySQL pour charger ce fichier dans MySQL comme suit :

mysql>SOURCE C:/wamp/www/sql/chap1/compagny.sql ; ← si vous êtes sous Windows il y aura un problème d'accent, pour l'insertion des données faire dans ce cas un copier coller ou utiliser la console de phpmyadmin.

pilot_id	certificate	name	number_flight	compagny
1	PL1	Antoine	100	AF
2	PL2	Jérôme	0	AF
3	PL3	Thierry	89	SI
4	PL4	Jean-Marie	20	AC
5	PL5	Pascale	30	BA
6	PL6	Cécile	120	AF
7	PL7	Naoudi	30	AF
8	PL8	Fenley	2	AF

Indication : pour une insertion multiple dans la table on utilisera la commande

INSERT INTO pilot (certificate, name, number_flight, compagny) VALUES (PL1, 'Antoine', 100, 'AF'), (PL2, 'Jérôme', 0, 'AF'), ... ;

DQL : TP

Interrogation des données SELECT

2 Exercice DDL de bases

A l'aide de la directive DISTINCT afficher les différentes compagnies.

3 Exercice

Sélectionner tous les pilotes dont l'id_pilot est supérieur à 4 strictement, puis à l'aide d'une autre requête, on sélectionnera le nom du pilote dont l'id_pilot est égal à 4.

Indication : SELECT col1[, col2,...] FROM nomTable [WHERE condition]

3.bis Exercice

Extraire les noms des pilotes et les noms des compagnies par ordre croissant puis décroissant de compagnie.

Puis extraire les noms des pilotes et les noms des compagnies par ordre croissant puis décroissant de compagnie et nom de pilote.

4 Exercice

Trouver le nombre de vols le plus important de la table Pilot. Puis le nombre de vol minimum. Et les deux dans la même pseud-table. On utilisera la clause MAX et MIN dans la partie SELECT sur les colonnes concernées.

5 Exercice (*)

Il est possible d'utiliser des sous-requêtes pour afficher des informations concernant par exemple le pilote ayant fait le plus de vols. Ainsi on peut écrire en SQL :

SELECT col1, col2 FROM ma_table WHERE col1=(SELECT [clause col1] FROM maTable) ;

On écrira donc la requête avec une sous requête permettant d'arriver au résultat suivant :

pilot_id	certificate	name	number_flight	compagny
6	PL6	Cécile	120	AF

6 Exercice (DDL)

On ajoutera les colonnes suivantes : bonus, type_plane, job_date dans la table Pilot. Écrire la commande DDL permettant de faire cela. Puis mettre à jour cette table à l'aide des lignes suivantes :

```
UPDATE Pilot SET bonus=100, type_plane='A380', job_date='2012-12-31 20:59:59' WHERE pilot_id>5 AND pilot_id<=7;
```

```
UPDATE Pilot SET bonus=300, type_plane='A320', job_date='2010-12-31 15:10:59' WHERE pilot_id<=5;
```

```
UPDATE Pilot SET bonus=500, type_plane='A340', job_date='2009-12-31 18:00:00' WHERE pilot_id=8;
```

7 Exercice

A l'aide de l'opérateur LIKE qui s'utilise avec la clause WHERE comme suit : WHERE col1 LIKE 'chaine' on utilisera les jockers suivants : % ou _, le premier jocker indique n'importe quel nombre de caractères le deuxième exactement un caractère. '%foo %' recherche un texte, une chaîne ayant le mot clef foo à l'intérieur.

Trouver tous les pilotes qui ont un « a » dans leur nom, tous les pilotes dont le nom est composé (signe -), tous les noms commençant par un « a » uniquement, tous les noms dont l'avant dernière lettre est un « l ».

Fonctions de groupe

Elles s'utilisent dans la clause SELECT sur une/des colonnes, elles permettent de regrouper des données,

Si vous utilisez les fonctions de groupement avec une requête ne contenant pas de clause GROUP BY, cela revient à grouper toutes les lignes.

AVG([DISTINCT] exp)	moyenne
COUNT({* DISTINCT] expr})	nombre de lignes
MAX([DISTINCT] exp)	max
MIN([DISTINCT] exp)	min
SUM([DISTINCT] exp)	somme
GROUP_CONCAT(exp)	composition d'un nombre de valeurs, concaténation de valeurs de champ a, b, c, d

8 Exercice

On donnera la moyenne des heures de vol et des primes(bonus) des pilotes de la compagnie 'AF'

9 Exercice

On donnera le nombre de primes (bonus) distincts de la table Pilot

10 Exercice (DQL)

On donnera les noms des pilotes de la compagnie AF

GROUP BY, HAVING

La clause GROUP BY s'applique au niveau surligné de l'instruction SQL ci-dessous, les colonnes présentes dans le SELECT doivent apparaître dans le GROUP BY, seul les fonctions ou expression peuvent exister en plus dans le SELECT. On ne peut pas utiliser d'alias dans la clause GROUP BY.

```
SELECT col1, col2, ..., [fonctions de groupement]
    [FROM tableName
      [WHERE where_definition]
      [GROUP BY {col1, col2, ...} [ASC | DESC], ...
      [HAVING where_definition]
    ]
```

Attention les alias ne marche pas sur les fonctions de regroupement. On essayera AVG(col1) 'moyenne'.

La clause WHERE permet d'exclure des lignes pour chaque groupement, ou rejeter des groupements entiers.

La clause GROUP BY liste des colonnes de groupement (les regroupe).

La clause HAVING permet de poser des conditions **sur chaque groupement** (sorte de filtre sur les données) elle ne s'utilise que sur une fonction d'agrégat (AVG, MAX, MIN, ...). **Attention, il faut bien comprendre que la clause WHERE agit avant HAVING qui fait un filtre sur un résultat après que la clause WHERE ait fait son travail.**

11 Exercice (*)

On donnera la moyenne des heures de vol et des primes pour chaque compagnie

DQL : TP

Interrogation des données SELECT

12 Exercice (*)

Le nombre d'heures de vol le plus élevé pour chaque compagnie.

Voici ce que MySQL va faire si on lui demande de regrouper les choses par compagnie par exemple à l'aide d'un GROUP BY sur compagny et de la fonction de regroupement MAX

1	PL1	Antoine	100	AF	300.00	A320	2010-12-31 15:10:59
2	PL2	Jérôme	0	AF	300.00	A320	2010-12-31 15:10:59
3	PL3	Thierry	89	SI	300.00	A320	2010-12-31 15:10:59
4	PL4	Jean-Marie	20	AC	300.00	A320	2010-12-31 15:10:59
5	PL5	Pascale	30	AC	300.00	A320	2010-12-31 15:10:59
6	PL6	Cécile	120	AF	100.00	A380	2012-12-31 20:59:59
7	PL7	Naoudi	30	AF	100.00	A380	2012-12-31 20:59:59
8	PL8	Fenley	2	AF	500.00	A340	2009-12-31 18:00:00

La pseudo-table sql donnera ceci, le nombre d'heures le plus élevé pour chaque compagnie :

AC	30
AF	120
SI	89

13 Exercice (*)

On crée la pseudo table qui donne la date d'embauche la plus récente pour chaque compagnie.

14 Exercice (*)

On donnera la pseudo table qui retourne le nombre de pilotes par type d'appareil et par compagnie, comme le certificat est unique on peut l'utiliser pour compter le nombre de pilotes.

15 Exercice (*)

La pseudo table qui affichera par compagnie les compagnies ayant plus d'un pilote (clause HAVING)

15.bis Exercice (*)

Combien d'avions par compagnie, et combien d'A320 par compagnie, combien d'avion différent par compagnie.

On aimerait savoir quels sont les types d'avions en commun que la compagnie AF et AC exploitent. En regardant la table on voit tout de suite que c'est A320.
En utilise pour faire cela une intersection entre deux ensembles de même type.

D'abord essayons de trouver tous les types d'avions que la compagnie AF utilise :

```
SELECT DISTINCT type_plane FROM Pilot WHERE compagny='AF' ;
```

Pour extraire les données que l'on souhaite il faut utiliser l'opérateur ensembliste IN

```
SELECT DISTINCT type_plane  
FROM Pilot WHERE compagny='AF' AND type_plane IN (SELECT type_plane FROM Pilot where compagny='AC');
```

type_plane
A320

Assez génial non ?

On pourrait également se demander maintenant quels sont les types d'avion que ces deux compagnies exploitent (c'est l'UNION ici)
16 Exercice (***)
Essayer en utilisant l'opérateur d'UNION qui élimine les doublons (sympa!) de répondre à cette question. Remarque l'opérateur UNION est commutatif.

Clef primaire et clef étrangère

La clef primaire

Elle peut être composée d'une ou plusieurs colonnes, elle permet d'identifier de manière unique une ligne de la table, **elle ne peut pas avoir la valeur «NULL», NB : une seule et unique clef primaire par table, celle-ci n'est pas obligatoire mais recommandée pour identifier de manière unique une ligne de la table.**

```
CREATE TABLE `article` (  
    `article_id` INT UNSIGNED AUTO_INCREMENT,  
    `title` TEXT,  
    PRIMARY KEY (`article_id`),  
) ENGINE=InnoDB AUTO_INCREMENT=1 ; <--- Dans ce cas la clef primaire s'auto-incrémente en partant de 1.
```

Une clef primaire est un index UNIQUE sur une colonne qui ne peut être NULL. Il n'est donc pas nécessaire d'ajouter la contrainte NOT NULL sur la/les colonne(s).

Supprimer une clef primaire

```
ALTER TABLE `article` DROP PRIMARY KEY ;
```

La clef étrangère

Elle permet de ne pas insérer de données qui n'auraient pas de sens dans les tables. Elle permet l'intégrité des données.

Pour créer une clef secondaire on doit :

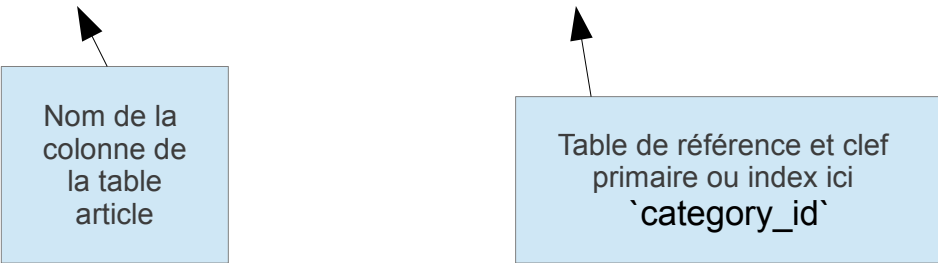
- choisir la ou les colonnes FOREIGN KEY
- choisir dans l'autre table la ou les colonnes qui va/vont servir de référence(s), REFERENCES

La clef secondaire ne peut s'ajouter dans la description d'une colonne, on met sa définition en général en dessous de la clef primaire.

Création d'une clef secondaire

```
CREATE TABLE `article` (  
  `article_id` INT UNSIGNED AUTO_INCREMENT,  
  `category` INT UNSIGNED, <--- doit avoir le même type que la clef primaire ou index de la table de référence  
  `title` TEXT,  
  PRIMARY KEY (`article_id`),  
  CONSTRAINT `fk_category_category_id` FOREIGN KEY (`category`) REFERENCES `category` (`category_id`)  
) ENGINE=InnoDB AUTO_INCREMENT=1 ;
```

```
CREATE TABLE `category` (  
  `category_id` INT UNSIGNED AUTO_INCREMENT,  
  `title` TEXT,  
  PRIMARY KEY (`category_id`),  
) ENGINE=InnoDB AUTO_INCREMENT=1 ;
```



Nom de la
colonne de
la table
article

Table de référence et clef
primaire ou index ici
`category_id`

Supprimer la clef secondaire sans supprimer la colonne associée à la clef :

```
ALTER TABLE `article` DROP FOREIGN KEY `fk_category_category_id` ;
```

On peut également ajouter une clef secondaire à une table existante :

```
ALTER TABLE `article` ADD CONSTRAINT `fk_category_category_id` FOREIGN KEY (`category`) REFERENCES  
`category` (`category_id`) ;
```

Changer la définition de la clef secondaire :

```
ALTER TABLE `article` DROP FOREIGN KEY `fk_category_category_id` ;  
ALTER TABLE `article` ADD CONSTRAINT `fk_category_category_id` FOREIGN KEY (`category`) REFERENCES  
`category` (`category_id`) ;
```

Si on a créé une clef secondaire « category » dans la table « article » qui référence la clef primaire de la table « category », on a les contraintes suivantes :

- on ne peut insérer un article dans la table « article » qu'avec un « ID » de catégorie existant
- on ne peut pas supprimer une catégorie qui a des références dans la table « article »

Options des clefs étrangères :

On a deux types DELETE et UPDATE

ON DELETE [RESTRICT| NO ACTION | SET NULL | CASCADE] permet de déterminer le comportement de MySQL en cas de suppression d'une référence

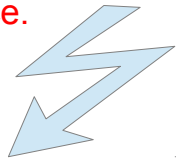
ON UPDATE [RESTRICT| NO ACTION | SET NULL | CASCADE] permet de déterminer le comportement de MySQL en cas de modification d'une référence.

ON DELETE RESTRICT et NO ACTION comportement par défaut, ces deux options on le même effet dans MySQL uniquement mais non pas dans les autres SGDB.

ON DELETE SET NULL dans ce cas NULL est substitué aux valeurs dont la référence est supprimée.

Exemple : si on a ON DELETE SET NULL sur la clef secondaire de la table « article », et si on supprime une catégorie alors les articles qui avaient cette référence (id de la catégorie supprimé) ne sont pas supprimés et la valeur NULL est mise à la place de l'id de la catégorie dans la table « article ».

ON DELETE CASCADE comportement plus risqué, plus violent ! Elle supprime toutes les lignes. Par exemple si on supprime une catégorie dans la table « category » de l'exemple précédent on supprimera tous les articles qui avaientt cette référence.



UPDATE

RESTRICT et NO ACTION : empêche la modification si elle casse la contrainte (comportement par défaut).

SET NULL : met NULL partout où la valeur modifiée était référencée.

CASCADE : modifie également la valeur là où elle est référencée.



Attention, ce n'est pas une bonne idée la valeur d'une clef primaire dans une table. Dans l'exemple ci-dessus, si on modifie l'ID d'une catégorie avec un UPDATE CASCADE cela modifie toutes les références à cette ID dans la table « article ».

Violation des contraintes d'unicité sans message d'erreur

On a en MySQL des commandes qui permettent d'ignorer les contraintes d'intégrités sans faire de modification dans les tables, il n'y a pas de message d'erreur renvoyé par MySQL:

IGNORE

En supposant que la clef secondaire client pointe sur la clef primaire client_id de la table client et que 7 n'existe pas

```
INSERT IGNORE INTO `command` (`name`, `code`, `client`) VALUES ('foo', 7, 'AA4');
```

Query OK, 0 rows affected (0.01 sec)

Cette commande marche également avec UPDATE.

Jointure

Les jointures permettent d'extraire des données de plusieurs tables, dans le modèle relationnel SQL on décompose en tables (entités) les données, il est donc nécessaire de faire des jointures pour extraire des données se trouvant dans différentes tables.

Principe :

Une jointure met en relation deux tables sur la base d'une clause de jointure (comparaison de colonne). Généralement, cette comparaison fait intervenir une clé étrangère d'une table avec une clé primaire d'une autre table. FK=PK, ils sont en particulier des indexes donc la requête s'effectue plus vite...

Le jointures interne (INNER JOIN)

Équijointure dans la clause de jointure on utilise « = »

Auto-jointure une équijointure sur la même table (on doit utiliser des alias pour distinguer les deux mêmes tables, voir exemple plus loin)

Dans une jointure interne seules les correspondances entre les deux tables sont extraites, on perd donc les données n'ayant pas de correspondances !

La jointure externe (LEFT OUTER JOIN ou RIGHT OUTER JOIN)

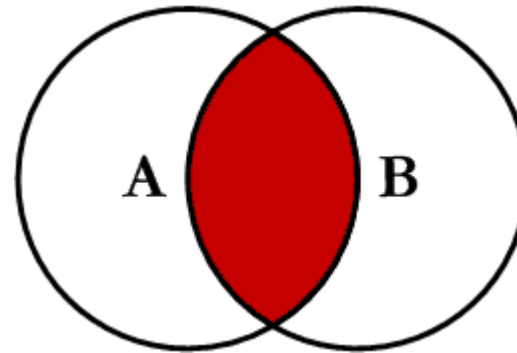
Elle favorise une table dite dominante par rapport à une table dite dominée, lorsqu'il n'y a pas de correspondance dans le prédicat de jointure les données apparaissent dans la pseudo-table.

Lorsqu'on extrait les données dans une jointure externe ce sont les données de la table dominante qui sortent avec des correspondances « NULL » si aucune correspondance n'est trouvée dans la table dominée.

Attention pas de FULL OUTER dans MySQL qui permet d'ignorer l'ordre entre les tables (subordonnée et dominante)

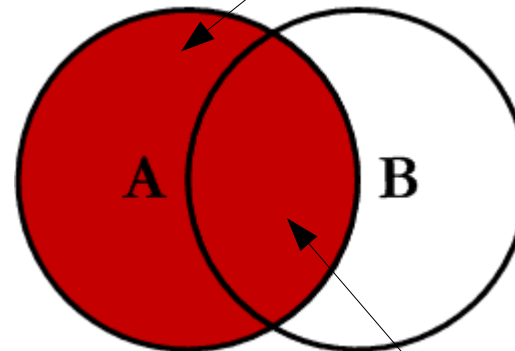
Jointure INNER JOIN, l'utilisation d'alias permet d'éviter les ambiguïtés entre les colonnes des tables, ils sont parfois nécessaires.

```
SELECT t1.title, t2.title  
FROM A as t1  
INNER JOIN B as t2  
ON t1.k=t2.k
```



Jointure externe LEFT OUTER JOIN la table dominante

```
SELECT t1.title, t2.title  
FROM A as t1  
LEFT OUTER JOIN B as t2  
ON t1.k=t2.k
```

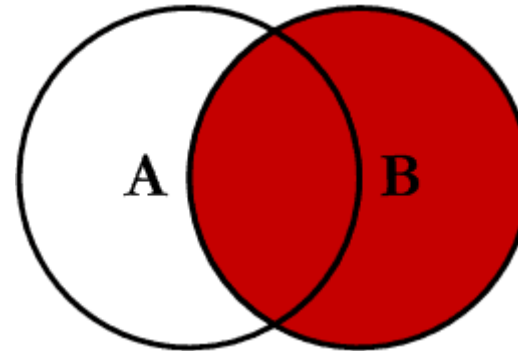


Ici toutes les correspondances
seront Null car la requête n'a rien trouvé
Dans la table B

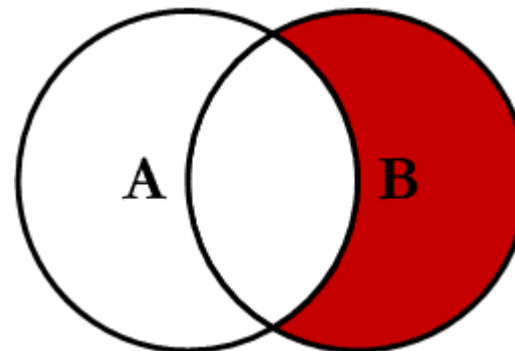
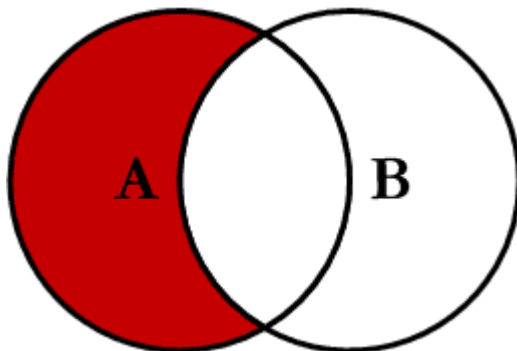
Il y a ici des correspondances

Jointure RIGHT OUTER JOIN la table dominante est B (petit b dans la requête)

```
SELECT t1.title, t2.title  
FROM A as t1  
RIGHT OUTER JOIN B as t2  
ON t1.k=t2.k
```



On peut s'amuser à exclure l'intersection pour les jointures externes en utilisant la restriction :
/*pour le dessin à gauche*/
SELECT t1.title, t2.title
FROM A as t1
LEFT OUTER JOIN B as t2
ON t1.k=t2.k
WHERE t2.k IS NULL



Exercice

On partira du code SQL suivant pour faire les exercices :

```
CREATE DATABASE IF NOT EXISTS
```

```
`db_shop`
```

```
DEFAULT CHARACTER
```

```
SET utf8
```

```
COLLATE utf8_general_ci;
```

```
USE `db_shop`;
```

```
CREATE TABLE `client` (
```

```
`client_id` INT UNSIGNED AUTO_INCREMENT,
```

```
`name` VARCHAR(30) NOT NULL,
```

```
PRIMARY KEY (`client_id`)
```

```
) ENGINE=InnoDB AUTO_INCREMENT=1;
```

```
CREATE TABLE `command` (
```

```
`command_id` INT UNSIGNED AUTO_INCREMENT,
```

```
`name` VARCHAR(30) NOT NULL,
```

```
`client` INT UNSIGNED , -- si on met NOT NULL la relation est 1 et si on ne met rien 0,1 la valeur NULL
```

```
`code` CHAR(3),
```

```
PRIMARY KEY (`command_id`),
```

```
CONSTRAINT `fk_client_client_id` FOREIGN KEY (`client`) REFERENCES `client` (`client_id`),
```

```
UNIQUE `un_code` (`code`)
```

```
) ENGINE=InnoDB AUTO_INCREMENT=1 ;
```

-- des commandes avec clients

```
INSERT INTO `client` (`name`) VALUES ('Antoine'), ('Cécile'), ('Fenley'), ('Naoudi');
```

```
INSERT INTO `command` (`name`, `code`, `client`) VALUES ('linux','AA4', 1), ('brompton', 'BB4', 1), ('portable', 'CC4', 2), ('sac  
à dos', 'EF5', 3);
```

-- une commande sans client ...

```
INSERT INTO `command` (`name`, `code`) VALUES ('linux','AA5');
```

-- un client sans commande

```
INSERT INTO `client` (`name`) VALUES ('Philippe');
```

Exercice 1

Ajouter une colonne « comment » à la table command. On a perdu la valeur « client_id » du client Naoudi comment lui ajouter un commentaire dans la table « commad », on ne fera qu'une seule requête UPDATE ?

Exercice 2

Que produit la requête suivante, écrire dans la console la commande :

```
INSERT INTO `command` (`name`, `code`, `client`) SELECT 'chaussettes', 'HG8', `client_id` as `client` FROM `client` WHERE `name`='Naoudi';
```

Exercices sur les jointures

Exercice 3

Afficher les clients qui on fait au moins une commande.

Exercice 4

Les clients ont-ils commandés des produits ou pas ?

Exercice 5

Toutes les commandes ont-elles un client ? Afficher tous les résultats.

Exercice 6

Trouver les clients dans les tables qui n'ont pas fait de commande.

Exercice 7

Isoler les commandes qui n'ont pas de client.

Exercice 8

On peut créer une vue dans la base de données db_shop à l'aide de la commande suivante :

```
CREATE VIEW nameView as SELECT ... ; C'est une table comme les autres, sauf qu'on ne pourra pas faire d'INSERT ni d'UPDATE.
```

Créer la vue des commandes avec le nom des clients et le code des produits commandés.

Indications, cette table persiste dans la base de données après la session en cours, si on veut supprimer la vue : DROP VIEW nameView ;

Introduction aux requêtes préparées

On a les variables utilisateurs valables uniquement lors d'une session, elles ne peuvent avoir que des valeurs numériques ou chaînes de caractères.

```
SET @foo=3 ;
```

```
SELECT @foo:= 5 ;
```

```
SELECT @foo ;
```

 affichera 5

Qu'est-ce qu'une requête préparée, c'est un modèle de requête que l'on enregistre. On l'appellera puis on lui passera des valeurs.

Par exemple si on fait souvent la requête suivante :

```
SELECT `name` FROM `client` WHERE client_id=1 ;
```

```
SELECT `name` FROM `client` WHERE client_id=2 ;
```

```
//...
```

On peut créer un modèle :

```
SELECT `name` FROM `client` WHERE client_id= ? ;
```

Tout comme les variables utilisateurs une requête préparée n'existe que pour la session en cours.

Syntaxe

```
PREPARE nameClient ← pas de guillemets
```

```
FROM 'SELECT `name` FROM `client` WHERE client_id= ? AND email = ? ;' ← entre guillemet
```

On peut passer des paramètres à la requête préparée à l'aide des variables utilisateurs :

```
SET @client_id=10 ;
```

```
SET @email='antoine.lucsko@wanadoo.fr' ;
```

```
EXECUTE nameClient USING @client_id, @email ;
```

Remarque importante, l'utilisation des requêtes préparées en SQL va en général de pair avec une API, comme avec PHP et PDO.

Exercice 9

Proposer une requête préparée. Si on ferme la session est-ce que cette requête préparée continue à exister dans notre base de données ?