III.3. Fonctions d'agrégation

Les fonctions d'agrégation, ou de groupement, sont des fonctions qui vont **regrouper les lignes**. Elles agissent sur une colonne, et renvoient un résultat unique pour toutes les lignes sélectionnées (ou pour chaque groupe de lignes, mais nous verrons cela plus tard).

Elles servent majoritairement à faire des **statistiques**, comme nous allons le voir dans la première partie de ce chapitre (compter des lignes, connaître une moyenne, trouver la valeur maximale d'une colonne,…). Nous verrons ensuite la fonction `GROUP_CONCAT()` qui, comme son nom l'indique, est une fonction de groupement qui sert à **concaténer** des valeurs.

III.3.1. Fonctions statistiques

La plupart des fonctions d'agrégation vont vous permettre de faire des statistiques sur vos données.

III.3.1.1. Nombre de lignes

La fonction `COUNT()` permet de savoir combien de lignes sont sélectionnées par la requête.

```sql
-- Combien de races avons-nous ? --
-- -------------------------------
SELECT COUNT(*) AS nb_races
FROM Race;

-- Combien de chiens avons-nous ? --
-- -------------------------------
SELECT COUNT(*) AS nb_chiens
FROM Animal
INNER JOIN Espece ON Espece.id = Animal.espece_id
WHERE Espece.nom_courant = 'Chien';
```

nb_races
8

nb_chiens

21

III.3.1.1.1. COUNT(*) ou COUNT(colonne)

Vous l'avez vu, j'ai utilisé `COUNT(*)` dans les exemples ci-dessus. Cela signifie que l'on compte tout simplement les lignes, sans se soucier de ce qu'elles contiennent.

Par contre, si on utilise `COUNT(colonne)`, seules les lignes dont la valeur de *colonne* n'est pas `NULL` seront prises en compte.

Exemple : comptons les lignes de la table *Animal*, avec `COUNT(*)` et `COUNT(race_id)`.

```
1 SELECT COUNT(race_id), COUNT(*)
2 FROM Animal;
```

COUNT(race_id)	COUNT(*)
31	60

Il n'y a donc que 31 animaux sur nos 60 pour lesquels la race est définie.

III.3.1.1.2. Doublons

Comme dans une requête `SELECT` tout à fait banale, il est possible d'utiliser le mot-clé `DISTINCT` pour ne pas prendre en compte les doublons.

Exemple : comptons le nombre de races distinctes définies dans la table *Animal*.

```
1 SELECT COUNT(DISTINCT race_id)
2 FROM Animal;
```

COUNT(DISTINCT race_id)
7

Parmi nos 31 animaux dont la race est définie, on trouve donc 7 races différentes.

III.3.1.2. Minimum et maximum

Nous avons déjà eu l'occasion de rencontrer la fonction `MIN(x)`, qui retourne la plus petite valeur de x. Il existe également une fonction `MAX(x)`, qui renvoie la plus grande valeur de x.

```sql
SELECT MIN(prix), MAX(prix)
FROM Race;
```

MIN(prix)	MAX(prix)
485.00	1235.00

Notez que `MIN()` et `MAX()` ne s'utilisent pas uniquement sur des données numériques. Si vous lui passez des chaînes de caractères, `MIN()` récupérera la première chaîne dans l'ordre alphabétique, `MAX()` la dernière ; avec des dates, `MIN()` renverra la plus vieille et `MAX()` la plus récente.

Exemple :

```sql
SELECT MIN(nom), MAX(nom), MIN(date_naissance), MAX(date_naissance)
FROM Animal;
```

MIN(nom)	MAX(nom)	MIN(date_naissance)	MAX(date_naissance)
Anya	Zonko	2006-03-15 14:26:00	2010-11-09 00:00:00

III.3.1.3. Somme et moyenne

III.3.1.3.1. Somme

La fonction `SUM(x)` renvoie la somme de x.

```sql
SELECT SUM(prix)
FROM Espece;
```

SUM(prix)
1200.00

III.3.1.3.2. Moyenne

La fonction `AVG(x)` (du mot anglais *average*) renvoie la valeur moyenne de x.

```sql
SELECT AVG(prix)
FROM Espece;
```

AVG(prix)
240.000000

III.3.2. Concaténation

III.3.2.1. Principe

Avec les fonctions d'agrégation, on regroupe plusieurs lignes. Les fonctions statistiques nous permettent d'avoir des informations fort utiles sur le résultat d'une requête, mais parfois, il est intéressant d'avoir également les valeurs concernées. Ceci est faisable avec `GROUP_CONCAT(nom_colonne)`. Cette fonction concatène les valeurs de *nom_colonne* pour chaque groupement réalisé.

Exemple : on récupère la somme des prix de chaque espèce, et on affiche les espèces concernées par la même occasion.

```sql
SELECT SUM(prix), GROUP_CONCAT(nom_courant)
FROM Espece;
```

SUM(prix)	GROUP_CONCAT(nom_courant)
1200.00	Chien,Chat,Tortue d'Hermann,Perroquet amazone,Rat brun

III.3.2.2. Syntaxe

Voici la syntaxe de cette fonction :

```sql
GROUP_CONCAT(
              [DISTINCT] col1 [, col2, ...]
              [ORDER BY col [ASC | DESC]]
              [SEPARATOR sep]
            )
```

— `DISTINCT` : sert comme d'habitude à éliminer les doublons.
— `col1` : est le nom de la colonne dont les valeurs doivent être concaténées. C'est le **seul argument obligatoire**.
— `col2,` : sont les éventuelles autres colonnes (ou chaînes de caractères) à concaténer.

— `ORDER BY` : permet de déterminer dans quel ordre les valeurs seront concaténées.
— `SEPARATOR` : permet de spécifier une chaîne de caractères à utiliser pour séparer les différentes valeurs. Par défaut, c'est une virgule.

III.3.2.3. Exemples

```sql
-- ------------------------------------
-- CONCATENATION DE PLUSIEURS COLONNES --
-- ------------------------------------
SELECT SUM(Race.prix), GROUP_CONCAT(Race.nom, Espece.nom_courant)
FROM Race
INNER JOIN Espece ON Espece.id = Race.espece_id;

-- ------------------------------------------------
-- CONCATENATION DE PLUSIEURS COLONNES EN PLUS JOLI --
-- ------------------------------------------------
SELECT SUM(Race.prix), GROUP_CONCAT(Race.nom, ' (',
    Espece.nom_courant, ')')
FROM Race
INNER JOIN Espece ON Espece.id = Race.espece_id;

-- --------------------------
-- ELIMINATION DES DOUBLONS --
-- --------------------------
SELECT SUM(Espece.prix), GROUP_CONCAT(DISTINCT Espece.nom_courant)
    -- Essayez sans le DISTINCT pour voir
FROM Espece
INNER JOIN Race ON Race.espece_id = Espece.id;

-- -------------------------
-- UTILISATION DE ORDER BY --
-- -------------------------
SELECT SUM(Race.prix), GROUP_CONCAT(Race.nom, ' (',
    Espece.nom_courant, ')' ORDER BY Race.nom DESC)
FROM Race
INNER JOIN Espece ON Espece.id = Race.espece_id;

-- ---------------------------
-- CHANGEMENT DE SEPARATEUR  --
-- ---------------------------
SELECT SUM(Race.prix), GROUP_CONCAT(Race.nom, ' (',
    Espece.nom_courant, ')' SEPARATOR ' - ')
FROM Race
INNER JOIN Espece ON Espece.id = Race.espece_id;
```

III.3.2.4. En résumé

— La plupart des fonctions d'agrégation permettent de faire des statistiques sur les données : nombre de lignes, moyenne d'une colonne,…
— `COUNT(*)` compte toutes les lignes quel que soit leur contenu, tandis que `COUNT(colonne_x)` compte les lignes pour lesquelles *colonne_x* n'est pas `NULL`.
— `GROUP_CONCAT(colonne_x)` permet de concaténer les valeurs de *colonne_x* dont les lignes ont été groupées par une autre fonction d'agrégation.

III.4. Regroupement

Vous savez donc que les fonctions d'agrégation groupent plusieurs lignes ensemble. Jusqu'à maintenant, toutes les lignes étaient chaque fois regroupées en une seule. Mais ce qui est particulièrement intéressant avec ces fonctions, c'est qu'il est possible de **regrouper les lignes en fonction d'un critère**, et d'avoir ainsi plusieurs groupes distincts. Par exemple, avec la fonction `COUNT(*)`, vous pouvez compter le nombre de lignes que vous avez dans la table *Animal*. Mais que diriez-vous de faire des groupes par espèces, et donc de savoir en une seule requête combien vous avez de chats, chiens, etc. ? Un simple groupement, et c'est fait !

Au programme de ce chapitre :

— les règles et la syntaxe à appliquer pour regrouper des lignes ;
— le groupement sur plusieurs critères ;
— les "super-agrégats" ;
— la sélection de certains groupes sur la base de critères.

III.4.1. Regroupement sur un critère

Pour regrouper les lignes selon un critère, il faut utiliser `GROUP BY`, qui se place après l'éventuelle clause `WHERE` (sinon directement après `FROM`), suivi du nom de la colonne à utiliser comme critère de regroupement.

```sql
SELECT ...
FROM nom_table
[WHERE condition]
GROUP BY nom_colonne;
```

Exemple 1 : comptons les lignes dans la table *Animal*, en regroupant sur le critère de l'espèce (donc avec la colonne *espece_id*).

```sql
SELECT COUNT(*) AS nb_animaux
FROM Animal
GROUP BY espece_id;
```

nb_animaux
21

20
15
4

Exemple 2 : Même chose, mais on ne prend que les mâles cette fois-ci.

```sql
SELECT COUNT(*) AS nb_males
FROM Animal
WHERE sexe = 'M'
GROUP BY espece_id;
```

nb_males
10
9
4
3

C'est déjà intéressant, mais nous n'allons pas en rester là. En effet, il serait quand même mieux de savoir à quelle espèce correspond quel nombre !

III.4.1.1. Voir d'autres colonnes

Pour savoir à quoi correspond chaque nombre, il suffit d'afficher également le critère qui a permis de regrouper les lignes. Dans notre cas, *espece_id*.

```sql
SELECT espece_id, COUNT(*) AS nb_animaux
FROM Animal
GROUP BY espece_id;
```

espece_id	nb_animaux
1	21
2	20
3	15
4	4

C'est déjà mieux, mais l'idéal serait d'avoir le nom des espèces directement. Qu'à cela ne tienne, il suffit de faire une jointure ! Sans oublier de changer le critère et de mettre *nom_courant* à la place de *espece_id*.

```sql
SELECT nom_courant, COUNT(*) AS nb_animaux
FROM Animal
INNER JOIN Espece ON Animal.espece_id = Espece.id
GROUP BY nom_courant;
```

nom_courant	nb_animaux
Chat	20
Chien	21
Perroquet amazone	4
Tortue d'Hermann	15

III.4.1.2. Colonnes sélectionnées

III.4.1.2.1. La règle SQL

Lorsque l'on fait un groupement dans une requête, avec `GROUP BY`, on ne peut sélectionner que deux types d'éléments dans la clause `SELECT` :

— une ou des colonnes **ayant servi de critère** pour le regroupement ;
— **une fonction d'agrégation** (agissant sur n'importe quelle colonne).

Cette règle est d'ailleurs logique. Imaginez la requête suivante :

```sql
SELECT nom_courant, COUNT(*) AS nb_animaux, date_naissance
FROM Animal
INNER JOIN Espece ON Animal.espece_id = Espece.id
GROUP BY nom_courant;
```

Que vient faire la date de naissance dans cette histoire ? Et surtout, quelle date de naissance espère-t-on sélectionner ? **Chaque ligne représente une espèce** puisque l'on a regroupé les lignes sur la base de *Espece.nom_courant*. Donc *date_naissance* n'a aucun sens par rapport aux groupes formés, une espèce n'ayant pas de date de naissance. Il en est de même pour les colonnes *sexe* ou *commentaires* par exemple.

Qu'en est-il des colonnes *Espece.id*, *Animal.espece_id*, ou *Espece.nom_latin* ? En groupant sur le nom courant de l'espèce, ces différentes colonnes ont un sens, et pourraient donc être utiles. Il a cependant été décidé que **par sécurité, la sélection de colonnes n'étant pas dans les critères de groupement serait interdite**. Cela afin d'éviter les situations comme au-dessus, où les colonnes sélectionnées n'ont aucun sens par rapport au groupement fait. Par conséquent,

si vous voulez afficher également l'id de l'espèce et son nom latin, il vous faudra grouper sur les trois colonnes : *Espece.nom_latin*, *Espece.nom_courant* et *Espece.id*. Les groupes créés seront les mêmes qu'en groupant uniquement sur *Espece.nom_courant*, mais votre requête respectera les standards SQL.

```
SELECT Espece.id, nom_courant, nom_latin, COUNT(*) AS nb_animaux
FROM Animal
INNER JOIN Espece ON Animal.espece_id = Espece.id
GROUP BY nom_courant, Espece.id, nom_latin;
```

id	nom_courant	nom_latin	nb_animaux
2	Chat	Felis silvestris	20
1	Chien	Canis canis	21
4	Perroquet amazone	Alipiopsitta xanthops	4
3	Tortue d'Hermann	Testudo hermanni	15

III.4.1.2.2. Le cas MySQL

On ne le répétera jamais assez : MySQL est un SGBD extrêmement permissif. Dans certains cas, c'est bien pratique, mais c'est toujours dangereux. Et notamment en ce qui concerne `GROUP BY`, MySQL ne sera pas perturbé pour un sou si vous sélectionnez une colonne qui n'est pas dans les critères de regroupement. Reprenons la requête qui sélectionne la colonne *date_naissance* alors que le regroupement se fait sur la base de l'*espece_id*. J'insiste, cette requête ne respecte pas la norme SQL, et n'a aucun sens. La plupart des SGBD vous renverront une erreur si vous tentez de l'exécuter.

```
SELECT nom_courant, COUNT(*) AS nb_animaux, date_naissance
FROM Animal
INNER JOIN Espece ON Animal.espece_id = Espece.id
GROUP BY espece_id;
```

Pourtant, loin de rouspéter, MySQL donne le résultat suivant :

nom_courant	nb_animaux	date_naissance
Chat	20	2010-03-24 02:23:00
Chien	21	2010-04-05 13:43:00
Perroquet amazone	4	2007-03-04 19:36:00
Tortue d'Hermann	15	2009-08-03 05:12:00

MySQL a tout simplement pris n'importe quelle valeur parmi celles du groupe pour la date de naissance. D'ailleurs, il est tout à fait possible que vous ayez obtenu des valeurs différentes des miennes. Soyez donc très prudents lorsque vous utilisez `GROUP BY`. Vous faites peut-être des requêtes qui n'ont aucun sens, et MySQL ne vous en avertira pas !

III.4.1.3. Tri des données

Par défaut dans MySQL, les données sont **triées sur la base du critère de regroupement**. C'est la raison pour laquelle, dans la requête précédente, la colonne *nom_courant* est triée par ordre alphabétique : c'est le premier critère de regroupement. MySQL permet d'utiliser les mots-clés `ASC` et `DESC` dans une clause `GROUP BY` pour choisir un tri ascendant (par défaut) ou descendant.

```
SELECT Espece.id, nom_courant, nom_latin, COUNT(*) AS nb_animaux
FROM Animal
INNER JOIN Espece ON Animal.espece_id = Espece.id
GROUP BY nom_courant DESC, Espece.id, nom_latin;  -- On trie par
    ordre anti-alphabétique sur le nom_courant
```

id	nom_courant	nom_latin	nb_animaux
3	Tortue d'Hermann	Testudo hermanni	15
4	Perroquet amazone	Alipiopsitta xanthops	4
1	Chien	Canis canis	21
2	Chat	Felis silvestris	20

Mais rien n'empêche d'utiliser une clause `ORDER BY` classique, après la clause `GROUP BY`. L'`ORDER BY` sera **prioritaire** sur l'ordre défini par la clause de regroupement.

Dans le même ordre d'idées, il n'est possible de faire un tri des données qu'à partir d'une colonne qui fait partie des critères de regroupement, ou à partir d'une fonction d'agrégation. Ça n'a pas plus de sens de trier les espèces par date de naissance que de sélectionner une date de naissance par espèce.

Vous pouvez par contre parfaitement écrire ceci :

```
SELECT Espece.id, nom_courant, nom_latin, COUNT(*) AS nb_animaux
FROM Animal
INNER JOIN Espece ON Animal.espece_id = Espece.id
GROUP BY nom_courant, Espece.id, nom_latin
ORDER BY nb_animaux;
```

id	nom_courant	nom_latin	nb_animaux

4	Perroquet amazone	Alipiopsitta xanthops	4
3	Tortue d'Hermann	Testudo hermanni	15
2	Chat	Felis silvestris	20
1	Chien	Canis canis	21

Notez que la norme SQL veut que l'on n'utilise pas d'expressions (fonction, opération mathématique,…) dans `GROUP BY` ou `ORDER BY`. C'est la raison pour laquelle j'ai mis `ORDER BY nb_animaux` et non pas `ORDER BY COUNT(*)`, bien qu'avec MySQL les deux fonctionnent. Pensez donc à utiliser des alias pour ces situations.

III.4.1.4. Et les autres espèces ?

La requête suivante nous donne le nombre d'animaux qu'on possède pour chaque espèce **dont on possède au moins un animal**. Comment peut-on faire pour afficher également les autres espèces ?

```
1 SELECT Espece.nom_courant, COUNT(*) AS nb_animaux
2 FROM Animal
3 INNER JOIN Espece ON Animal.espece_id = Espece.id
4 GROUP BY nom_courant;
```

Essayons donc avec une jointure externe, puisqu'il faut tenir compte de toutes les espèces, même celles qui n'ont pas de correspondance dans la table *Animal*.

```
1 SELECT Espece.nom_courant, COUNT(*) AS nb_animaux
2 FROM Animal
3 RIGHT JOIN Espece ON Animal.espece_id = Espece.id -- RIGHT puisque
      la table Espece est à droite.
4 GROUP BY nom_courant;
```

nom_courant	nb_animaux
Chat	20
Chien	21
Perroquet amazone	4
Rat brun	1
Tortue d'Hermann	15

Les rats bruns apparaissent bien. En revanche, ce n'est pas 1 qu'on attend, mais 0, puisqu'on n'a pas de rats dans notre élevage. Cela dit, ce résultat est logique : avec la jointure externe, on aura une ligne correspondant aux rats bruns, avec `NULL` dans toutes les colonnes de la table *Animal*. Donc ce qu'il faudrait, c'est avoir les cinq espèces, mais ne compter que lorsqu'il y a un animal correspondant. Pour ce faire, il suffit de faire `COUNT(Animal.espece_id)` par exemple.

```sql
SELECT Espece.nom_courant, COUNT(Animal.espece_id) AS nb_animaux
FROM Animal
RIGHT JOIN Espece ON Animal.espece_id = Espece.id
GROUP BY nom_courant;
```

nom_courant	nb_animaux
Chat	20
Chien	21
Perroquet amazone	4
Rat brun	0
Tortue d'Hermann	15

C'est pas magique ça ? 🥳

III.4.2. Regroupement sur plusieurs critères

J'ai mentionné le fait qu'il était possible de grouper sur plusieurs colonnes, mais jusqu'à maintenant, cela n'a servi qu'à pouvoir afficher correctement les colonnes voulues, sans que ça n'influe sur les groupes. On n'avait donc en fait qu'un seul critère, représenté par plusieurs colonnes. Voyons maintenant un exemple avec deux critères différents (qui ne créent pas les mêmes groupes).

Les deux requêtes suivantes permettent de savoir combien d'animaux de chaque espèce vous avez dans la table *Animal*, ainsi que combien de mâles et de femelles, toutes espèces confondues.

```sql
SELECT nom_courant, COUNT(*) as nb_animaux
FROM Animal
INNER JOIN Espece ON Espece.id = Animal.espece_id
GROUP BY nom_courant;

SELECT sexe, COUNT(*) as nb_animaux
FROM Animal
GROUP BY sexe;
```

nom_courant	nb_animaux
Chat	20
Chien	21
Perroquet amazone	4
Tortue d'Hermann	15

sexe	nb_animaux
NULL	3
F	31
M	26

En faisant un regroupement multicritère, il est possible de savoir facilement combien de mâles et de femelles **par espèce** il y a dans la table *Animal*. Notez que **l'ordre des critères a son importance**.

Exemple 1 : on regroupe d'abord sur l'espèce, puis sur le sexe.

```
SELECT nom_courant, sexe, COUNT(*) as nb_animaux
FROM Animal
INNER JOIN Espece ON Espece.id = Animal.espece_id
GROUP BY nom_courant, sexe;
```

nom_courant	sexe	nb_animaux
Chat	NULL	2
Chat	F	9
Chat	M	9
Chien	F	11
Chien	M	10
Perroquet amazone	F	1
Perroquet amazone	M	3
Tortue d'Hermann	NULL	1
Tortue d'Hermann	F	10
Tortue d'Hermann	M	4

Exemple 2 : on regroupe d'abord sur le sexe, puis sur l'espèce.

```
SELECT nom_courant, sexe, COUNT(*) as nb_animaux
FROM Animal
INNER JOIN Espece ON Espece.id = Animal.espece_id
GROUP BY sexe,nom_courant;
```

nom_courant	sexe	nb_animaux
Chat	NULL	2
Tortue d'Hermann	NULL	1
Chat	F	9
Chien	F	11
Perroquet amazone	F	1
Tortue d'Hermann	F	10
Chat	M	9
Chien	M	10
Perroquet amazone	M	3
Tortue d'Hermann	M	4

Étant donné que le regroupement par sexe donnait trois groupes différents, et le regroupement par espèce donnait quatre groupes différents, il peut y avoir jusqu'à douze (3 x 4) groupes lorsque l'on regroupe en se basant sur les deux critères. Ici, il y en aura moins puisque le sexe de tous les chiens et de tous les perroquets est défini (pas de `NULL`).

III.4.3. Super-agrégats

Parlons maintenant de l'option `WITH ROLLUP` de `GROUP BY`. Cette option va afficher des lignes supplémentaires dans la table de résultats. Ces lignes représenteront des "super-groupes" (ou super-agrégats), donc des "groupes de groupes". Deux petits exemples, et vous aurez compris !

III.4.3.0.1. Exemple avec un critère de regroupement

```
SELECT nom_courant, COUNT(*) as nb_animaux
FROM Animal
INNER JOIN Espece ON Espece.id = Animal.espece_id
GROUP BY nom_courant WITH ROLLUP;
```

nom_courant	nb_animaux
Chat	20
Chien	21
Perroquet amazone	4
Tortue d'Hermann	15
NULL	60

Nous avons donc 20 chats, 21 chiens, 4 perroquets et 15 tortues. Et combien font 20 + 21 + 4 + 15 ? 60 ! Exactement. La ligne supplémentaire représente donc le regroupement de nos quatre groupes basé sur le critère `GROUP BY nom_courant`.

III.4.3.0.2. Exemple avec deux critères de regroupement

```sql
SELECT nom_courant, sexe, COUNT(*) as nb_animaux
FROM Animal
INNER JOIN Espece ON Espece.id = Animal.espece_id
WHERE sexe IS NOT NULL
GROUP BY nom_courant, sexe WITH ROLLUP;
```

nom_courant	sexe	nb_animaux
Chat	F	9
Chat	M	9
Chat	NULL	18
Chien	F	11
Chien	M	10
Chien	NULL	21
Perroquet amazone	F	1
Perroquet amazone	M	3
Perroquet amazone	NULL	4
Tortue d'Hermann	F	10
Tortue d'Hermann	M	4
Tortue d'Hermann	NULL	14
NULL	NULL	57

Les deux premières lignes correspondent aux nombres de chats mâles et femelles. Jusque-là, rien de nouveau. Par contre, la troisième ligne est une ligne insérée par `WITH ROLLUP`, et contient le nombre de chats (mâles et femelles). Nous avons fait des groupes en séparant les espèces et les sexes, et `WITH ROLLUP` a créé des "super-groupes" en regroupant les sexes mais gardant les espèces séparées. Nous avons donc également le nombre de chiens à la sixième ligne, de perroquets à la neuvième, et de tortues à la douzième. Quant à la toute dernière ligne, c'est un "super-super-groupe" qui réunit tous les groupes ensemble.

C'est en utilisant `WITH ROLLUP` qu'on se rend compte que l'ordre des critères est vraiment important. En effet, voyons ce qui se passe si on échange les deux critères *nom_courant* et *sexe*.

```
SELECT nom_courant, sexe, COUNT(*) as nb_animaux
FROM Animal
INNER JOIN Espece ON Espece.id = Animal.espece_id
WHERE sexe IS NOT NULL
GROUP BY sexe, nom_courant WITH ROLLUP;
```

nom_courant	sexe	nb_animaux
Chat	F	9
Chien	F	11
Perroquet amazone	F	1
Tortue d'Hermann	F	10
NULL	F	31
Chat	M	9
Chien	M	10
Perroquet amazone	M	3
Tortue d'Hermann	M	4
NULL	M	26
NULL	NULL	57

Cette fois-ci, les super-groupes ne correspondent pas aux espèces, mais aux sexes, c'est-à-dire au premier critère. Le regroupement se fait bien dans l'ordre donné par les critères.

J'ai ajouté la condition `WHERE sexe IS NOT NULL` pour des raisons de lisibilité uniquement, étant donné que sans cette condition, d'autres `NULL` seraient apparus, rendant plus compliquée l'explication des super-agrégats.

III.4.3.0.3. NULL, c'est pas joli

Il est possible d'éviter d'avoir ces NULL dans les lignes des super-groupes. Pour cela, on peut utiliser la fonction COALESCE(). Cette fonction prend autant de paramètres que l'on veut, et renvoie le premier paramètre non `NULL`.

Exemples :

```
SELECT COALESCE(1, NULL, 3, 4); -- 1
SELECT COALESCE(NULL, 2);       -- 2
SELECT COALESCE(NULL, NULL, 3); -- 3
```

Voici comment l'utiliser dans le cas des super-agrégats.

```
SELECT COALESCE(nom_courant, 'Total'), COUNT(*) as nb_animaux
FROM Animal
INNER JOIN Espece ON Espece.id = Animal.espece_id
GROUP BY nom_courant WITH ROLLUP;
```

COALESCE(nom_courant, 'Total')	nb_animaux
Chat	20
Chien	21
Perroquet amazone	4
Tortue d'Hermann	15
Total	60

Tant qu'il s'agit de simples groupes, *nom_courant* contient bien le nom de l'espèce. `COALESCE()` renvoie donc celui-ci. Par contre, quand il s'agit du super-groupe, la colonne *nom_courant* du résultat contient `NULL`, et donc `COALESCE()` va renvoyer "Total".

Si vous utilisez `COALESCE()` dans ce genre de situation, il est impératif que vos critères de regroupement ne contiennent pas `NULL` (ou que vous éliminiez ces lignes-là). Sinon, vous aurez "Total" à des lignes qui ne sont pas des super-groupes.

Exemple : groupons sur le sexe, sans éliminer les lignes pour lesquelles le sexe n'est pas défini.

```
SELECT COALESCE(sexe, 'Total'), COUNT(*) as nb_animaux
FROM Animal
INNER JOIN Espece ON Espece.id = Animal.espece_id
```

```
4 GROUP BY sexe WITH ROLLUP;
```

COALESCE(sexe, 'Total')	nb_animaux
Total	3
F	31
M	26
Total	60

III.4.4. Conditions sur les fonctions d'agrégation

Il n'est pas possible d'utiliser la clause `WHERE` pour faire des conditions sur une fonction d'agrégation. Donc, si l'on veut afficher les espèces dont on possède plus de 15 individus, la requête suivante ne fonctionnera pas.

```
1 SELECT nom_courant, COUNT(*)
2 FROM Animal
3 INNER JOIN Espece ON Espece.id = Animal.espece_id
4 WHERE COUNT(*) > 15
5 GROUP BY nom_courant;
```

```
1 ERROR 1111 (HY000): Invalid use of group function
```

Il faut utiliser une clause spéciale : `HAVING`. Cette clause se place juste après le `GROUP BY`.

```
1 SELECT nom_courant, COUNT(*)
2 FROM Animal
3 INNER JOIN Espece ON Espece.id = Animal.espece_id
4 GROUP BY nom_courant
5 HAVING COUNT(*) > 15;
```

nom_courant	COUNT(*)
Chat	20
Chien	21

Il est également possible d'utiliser un alias dans une condition `HAVING` :

```sql
SELECT nom_courant, COUNT(*) as nombre
FROM Animal
INNER JOIN Espece ON Espece.id = Animal.espece_id
GROUP BY nom_courant
HAVING nombre > 15;
```

nom_courant	nombre
Chat	20
Chien	21

III.4.4.1. Optimisation

Les conditions données dans la clause `HAVING` ne doivent pas nécessairement comporter une fonction d'agrégation. Les deux requêtes suivantes donneront par exemple des résultats équivalents :

```sql
SELECT nom_courant, COUNT(*) as nombre
FROM Animal
INNER JOIN Espece ON Espece.id = Animal.espece_id
GROUP BY nom_courant
HAVING nombre > 6 AND SUBSTRING(nom_courant, 1, 1) = 'C'; -- Deux
    conditions dans HAVING

SELECT nom_courant, COUNT(*) as nombre
FROM Animal
INNER JOIN Espece ON Espece.id = Animal.espece_id
WHERE SUBSTRING(nom_courant, 1, 1) = 'C'                  -- Une
    condition dans WHERE
GROUP BY nom_courant
HAVING nombre > 6;                                        -- Et une
    dans HAVING
```

Il est cependant préférable, et de loin, d'utiliser la clause `WHERE` autant que possible, c'est-à-dire pour toutes les conditions, sauf celles utilisant une fonction d'agrégation. En effet, les conditions `HAVING` ne sont absolument pas optimisées, au contraire des conditions `WHERE`.

III.4.4.2. En résumé

— Utiliser `GROUP BY` en combinaison avec une fonction d'agrégation permet de regrouper les lignes selon un ou plusieurs critères.

— Si l'on groupe sur un seul critère, on aura autant de groupes (donc de lignes) que de valeurs différentes dans la colonne utilisée comme critère.
— Si l'on groupe sur plusieurs critères, le nombre maximum de groupes résultant est obtenu en multipliant le nombre de valeurs différentes dans chacune des colonnes utilisées comme critère.
— Selon la norme SQL, si l'on utilise `GROUP BY`, on ne peut avoir dans la clause `SELECT` que des fonctions d'agrégation, ou des colonnes utilisées comme critère dans le `GROUP BY`.
— L'option `WITH ROLLUP` permet de créer des super-groupes (ou groupes de groupe).
— Une condition sur le résultat d'une fonction d'agrégation se place dans une clause `HAVING`, et non dans la clause `WHERE`.

III.5. Exercices sur les agrégats

Jusqu'à maintenant, tout a été très théorique. Or, la meilleure façon d'apprendre, c'est la pratique. Voici donc quelques exercices que je vous conseille de faire. S'il vaut mieux que vous essayiez par vous-mêmes avant de regarder la solution, ne restez cependant pas bloqués trop longtemps sur un exercice, et prenez toujours le temps de bien comprendre la solution.

III.5.1. Du simple...

III.5.1.1. 1. Combien de races avons-nous dans la table *Race* ?

Contenu masqué n°21

III.5.1.2. 2. De combien de chiens connaissons-nous le père ?

Contenu masqué n°22

III.5.1.3. 3. Quelle est la date de naissance de notre plus jeune femelle ?

Contenu masqué n°23

III.5.1.4. 4. En moyenne, quel est le prix d'un chien ou d'un chat de race, par espèce, et en général ?

Contenu masqué n°24

III.5.1.5. 5. Combien avons-nous de perroquets mâles et femelles, et quels sont leurs noms (en une seule requête bien sûr) ?

Contenu masqué n°25

III.5.2. ...Vers le complexe

III.5.2.1. 1. Quelles sont les races dont nous ne possédons aucun individu ?

Contenu masqué n°26

III.5.2.2. 2. Quelles sont les espèces (triées par ordre alphabétique du nom latin) dont nous possédons moins de cinq mâles ?

Contenu masqué n°27

III.5.2.3. 3. Combien de mâles et de femelles de chaque race avons-nous, avec un compte total intermédiaire pour les races (mâles et femelles confondues) et pour les espèces ? Afficher le nom de la race, et le nom courant de l'espèce.

Contenu masqué n°28

III.5.2.4. 4. Quel serait le coût, par espèce et au total, de l'adoption de Parlotte, Spoutnik, Caribou, Cartouche, Cali, Canaille, Yoda, Zambo et Lulla ?

Petit indice, pour avoir le prix d'un animal selon que sa race soit définie ou non, vous pouvez utiliser une fonction que nous venons de voir au chapitre précédent.

Contenu masqué n°29

Et voilà pour les nombres et les chaînes de caractères ! Notez que les fonctions d'agrégat sont parmi les plus utilisées donc soyez bien sûrs d'avoir compris comment elles fonctionnent, couplées à `GROUP BY` ou non.

Contenu masqué

Contenu masqué n°21

```sql
SELECT COUNT(*)
FROM Race;
```

Simple échauffement. Retourner au texte.

Contenu masqué n°22

```sql
SELECT COUNT(pere_id)
FROM Animal
INNER JOIN Espece ON Espece.id = Animal.espece_id
WHERE Espece.nom_courant = 'Chien';
```

L'astuce ici était de ne pas oublier de donner la colonne *pere_id* en paramètre à `COUNT()`, pour ne compter que les lignes où *pere_id* est non `NULL`. Si vous avez fait directement `WHERE espece_id = 1` au lieu d'utiliser une jointure pour sélectionner les chiens, ce n'est pas bien grave. Retourner au texte.

Contenu masqué n°23

```sql
SELECT MAX(date_naissance)
FROM Animal
WHERE sexe = 'F';
```

Retourner au texte.

Contenu masqué n°24

```sql
SELECT nom_courant AS Espece, AVG(Race.prix) AS prix_moyen
FROM Race
INNER JOIN Espece ON Race.espece_id = Espece.id
WHERE Espece.nom_courant IN ('Chat', 'Chien')
GROUP BY Espece.nom_courant WITH ROLLUP;
```

Ne pas oublier `WITH ROLLUP` pour avoir le résultat général. Retourner au texte.

Contenu masqué n°25

```sql
SELECT sexe, COUNT(*), GROUP_CONCAT(nom SEPARATOR ', ')
FROM Animal
INNER JOIN Espece ON Animal.espece_id = Espece.id
WHERE nom_courant = 'Perroquet amazone'
GROUP BY sexe;
```

Il suffisait de se souvenir de la méthode `GROUP_CONCAT()` pour pouvoir réaliser simplement cette requête. Peut-être avez-vous groupé sur l'espèce aussi (avec *nom_courant* ou autre). Ce n'était pas nécessaire puisqu'on avait restreint à une seule espèce avec la clause `WHERE`. Cependant, cela n'influe pas sur le résultat, mais sur la rapidité de la requête. Retourner au texte.

Contenu masqué n°26

```sql
SELECT Race.nom, COUNT(Animal.race_id) AS nombre
FROM Race
LEFT JOIN Animal ON Animal.race_id = Race.id
GROUP BY Race.nom
HAVING nombre = 0;
```

Il fallait ici ne pas oublier de faire une jointure externe (`LEFT` ou `RIGHT`, selon votre requête), ainsi que de mettre la colonne *Animal.race_id* (ou *Animal.id*, ou *Animal.espece_id* mais c'est moins intuitif) en paramètre de la fonction `COUNT()`. Retourner au texte.

Contenu masqué n°27

```sql
SELECT Espece.nom_latin, COUNT(espece_id) AS nombre
FROM Espece
LEFT JOIN Animal ON Animal.espece_id = Espece.id
WHERE sexe = 'M' OR Animal.id IS NULL
GROUP BY Espece.nom_latin
HAVING nombre < 5;
```

À nouveau, une jointure externe et *espece_id* en argument de `COUNT()`, mais il y avait ici une petite subtilité en plus. Puisqu'on demandait des informations sur les mâles uniquement, il fallait une condition `WHERE sexe = 'M'`. Mais cette condition fait que les lignes de la jointure provenant de la table *Espece* n'ayant aucune correspondance dans la table *Animal* sont éliminées également (puisque forcément, toutes les colonnes de la table *Animal*, dont *sexe*, seront à `NULL` pour ces lignes). Par conséquent, il fallait ajouter une condition permettant de garder ces fameuses lignes (les espèces pour lesquelles on n'a aucun individu, donc aucun mâle). Il fallait donc ajouter `OR Animal.id IS NULL`, ou faire cette condition sur toute autre colonne d'*Animal* ayant la contrainte `NOT NULL`, et qui donc ne sera `NULL` que lors d'une jointure externe, en cas de non-correspondance avec l'autre table. Il n'y a plus alors qu'à ajouter la clause `HAVING` pour sélectionner les espèces ayant moins de cinq mâles. Retourner au texte.

Contenu masqué n°28

```sql
SELECT Animal.sexe, Race.nom, Espece.nom_courant, COUNT(*) AS
    nombre
FROM Animal
INNER JOIN Espece ON Animal.espece_id = Espece.id
INNER JOIN Race ON Animal.race_id = Race.id
WHERE Animal.sexe IS NOT NULL
GROUP BY Espece.nom_courant, Race.nom, sexe WITH ROLLUP;
```

Deux jointures sont nécessaires pour pouvoir afficher les noms des races et des espèces. Il suffit alors de ne pas oublier l'option `WITH ROLLUP` et de mettre les critères de regroupement dans le bon ordre pour avoir les super-agrégats voulus. Retourner au texte.

Contenu masqué n°29

```sql
SELECT Espece.nom_courant, SUM(COALESCE(Race.prix, Espece.prix)) AS
    somme
FROM Animal
```

```
3 INNER JOIN Espece ON Espece.id = Animal.espece_id
4 LEFT JOIN Race ON Race.id = Animal.race_id
5 WHERE Animal.nom IN ('Parlotte', 'Spoutnik', 'Caribou',
      'Cartouche', 'Cali', 'Canaille', 'Yoda', 'Zambo', 'Lulla')
6 GROUP BY Espece.nom_courant WITH ROLLUP;
```

C'est ici la fonction `SUM()` qu'il fallait utiliser, puisqu'on veut le prix total par groupe. Sans oublier le `WITH ROLLUP` pour avoir également le prix total tous groupes confondus. Quant au prix de chaque animal, c'est typiquement une situation où l'on peut utiliser `COALESCE()` ! Retourner au texte.