BASES DE DONNÉES : OPÉRATIONS SUR UNE TABLE

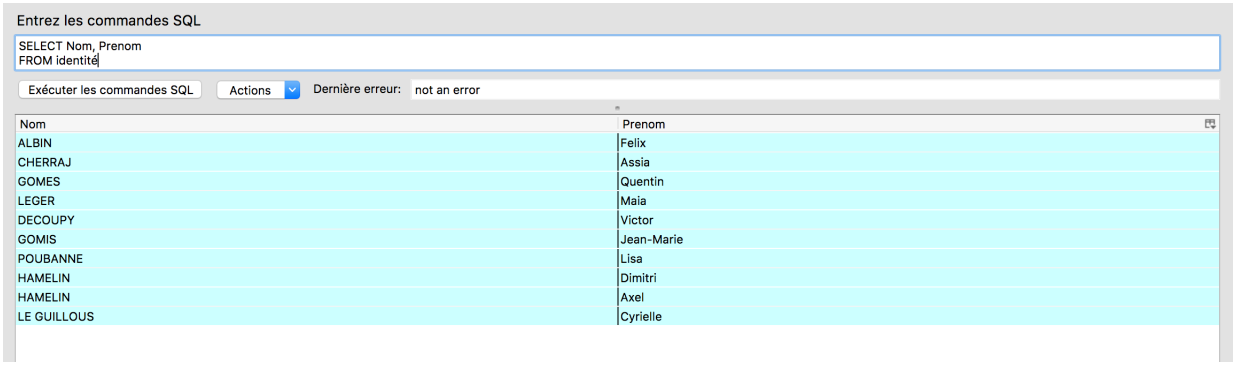
I Opérations ensemblistes

I.1 Projection

La projection est l'opération permettant de ne choisir que certains attributs (donc certaines colonnes) d'une relation. Par exemple lorsque l'on projette la relation *identité* en ne gardant que les attributs *Nom* et *Prénom*, la requête SQL est :

SQL

```
SELECT Nom, Prenom
FROM identité;
```



Cette requête s'écrit en algèbre relationnelle :

$$\pi_{Nom,Prenom}(\text{identité})$$

Remarque 2.1. *Le caractère " ;" permet de fermer une requête SQL.*

Pour éviter des redondances dans les résultats, on peut utiliser le mot-clé SELECT DISTINCT au lieu de SELECT.

La clause LIMIT est à utiliser dans une requête lorsqu'on souhaite spécifier le nombre maximum de lignes dans le résultat. Par exemple,

SQL

```
SELECT Nom, Prenom
FROM identité
LIMIT 4;
```

ne donnera que les 4 premières lignes du résultat précédent.

La clause OFFSET permet d'ignorer les premières lignes. Par exemple,

SQL

```
SELECT Nom, Prenom
FROM identité
LIMIT 4 OFFSET 3;
```

renverra les lignes de LEGER à POUBANNE (les trois premières lignes sont ignorées et on récupère 4 lignes maximum).

Enfin, le mot clé ORDER BY permet d'ordonner le résultat selon un attribut.

SQL

```
SELECT Nom, Prenom
FROM identité
ORDER BY Nom;
```

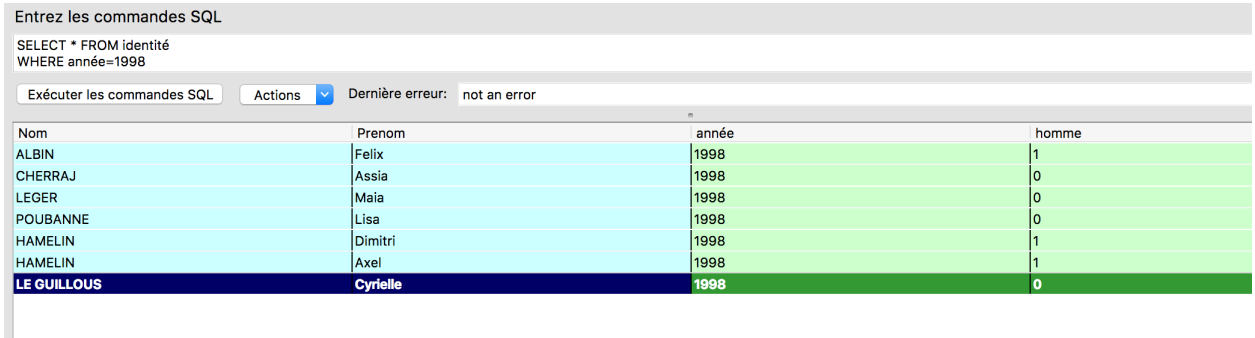
Les lignes seront ordonnées selon les noms des élèves.

I.2 Sélection

La **sélection** est l'opération permettant de ne choisir que les entrées (donc les lignes) d'une relation vérifiant une certaine condition. Par exemple si l'on cherche les élèves nés en 1998, la requête SQL est :

SQL

```
SELECT *
FROM identité
WHERE année=1998;
```



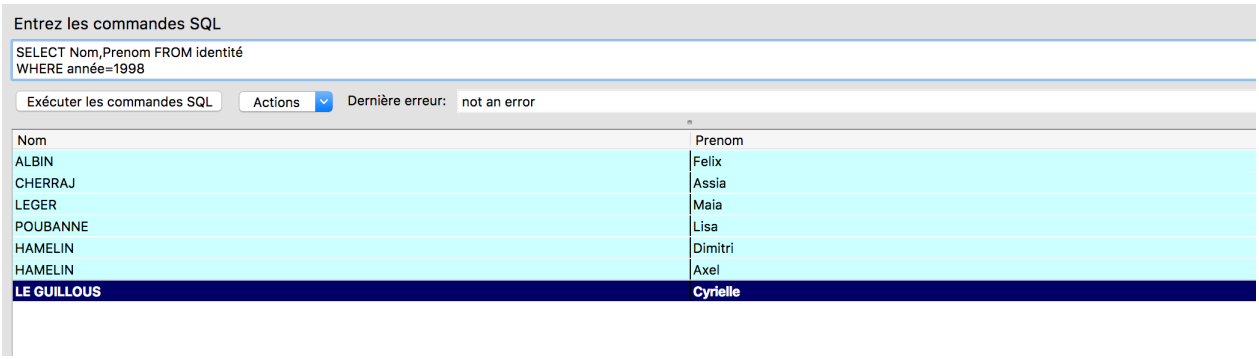
Cette requête s'écrit en algèbre relationnelle :

$$\sigma_{\text{année}=1998}(\text{identité})$$

On peut évidemment combiner projection et sélection :

SQL

```
SELECT Nom, Prenom
FROM identité
WHERE année=1998;
```



Cette requête s'écrit en algèbre relationnelle :

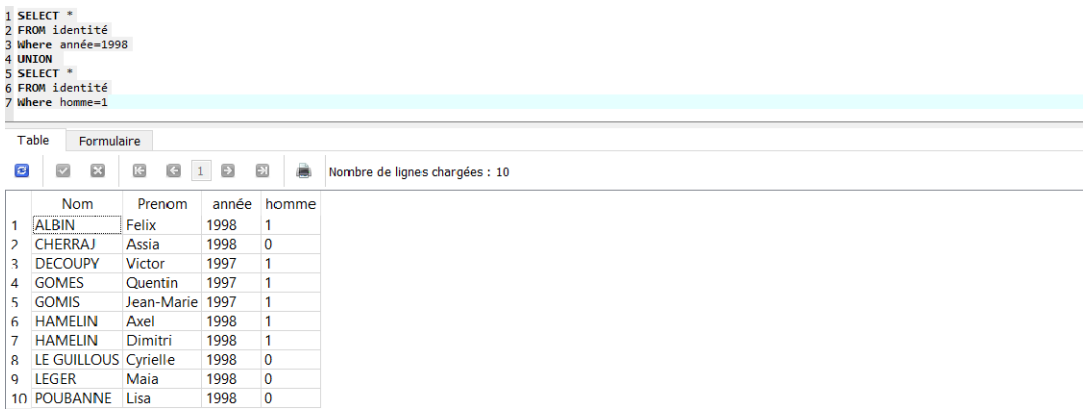
$$\pi_{Nom,Prenom}\left(\sigma_{\text{année}=1998}(\text{identité})\right)$$

I.3 Union

La réunion est l'opération algébrique classique. Lors d'une requête, les attributs projetés doivent être les mêmes. Exemple : si on recherche les élèves nés en 1998 ou les hommes :

SQL

```
SELECT *
FROM identité
WHERE année=1998
UNION
SELECT *
FROM identité
WHERE homme=1;
```



Cette requête s'écrit en algèbre relationnelle :

$$\sigma_{\text{année}=1998}(\text{identité}) \cup \sigma_{\text{homme}=1}(\text{identité})$$

Remarque 2.2. *On peut effectuer cette exemple autrement avec une sélection :*

SQL

```
SELECT *
FROM identité
WHERE année=1998 OR homme=1;
```

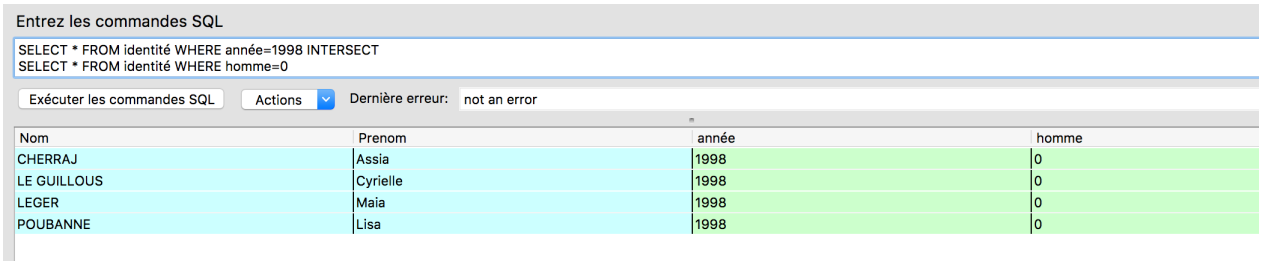
Cependant, on verra dans le chapitre suivant que la réunion peut s'effectuer entre tables différentes.

I.4 Intersection

L'intersection est l'opération algébrique classique. Lors d'une requête, les attributs projetés doivent être les mêmes. Exemple : si on recherche les élèves nés en 1998 qui sont des femmes :

SQL

```
SELECT *
FROM identité
WHERE année=1998
INTERSECT
SELECT *
FROM identité
WHERE homme=0;
```



Cette requête s'écrit en algèbre relationnelle :

$$\sigma_{\text{année}=1998}(\text{identité}) \cap \sigma_{\text{homme}=0}(\text{identité})$$

Remarque 2.3. *On peut effectuer cette exemple autrement avec une sélection :*

SQL

```
SELECT *
FROM identité
WHERE année=1998 AND homme=0;
```

Cependant, on verra dans le chapitre suivant que l'intersection peut s'effectuer entre tables différentes.

I.5 Différence ensembliste

La différence ensembliste est l'opération algébrique classique. Elle concerne les entrées uniquement. Lors d'une requête, les attributs projetés doivent être les mêmes. Exemple : si on recherche les élèves nés en 1998 qui ne sont pas des femmes :

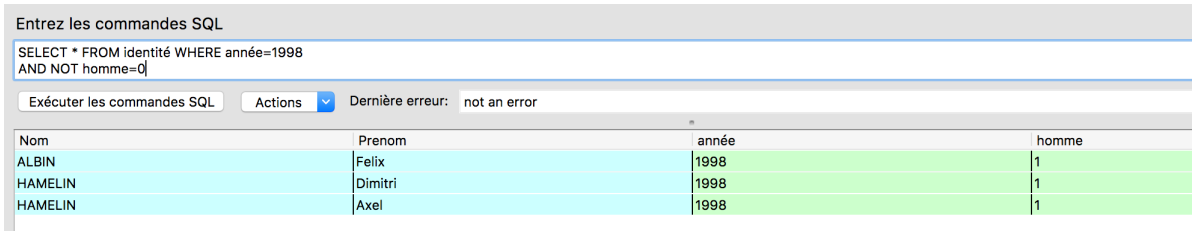
SQL

```
SELECT *
FROM identité
WHERE année=1998
EXCEPT
SELECT *
FROM identité
WHERE homme=0;
```

qu'on peut également obtenir avec :

SQL

```
SELECT *
FROM identité
WHERE année=1998
AND NOT homme=0;
```



Cette requête s'écrit en algèbre relationnelle :

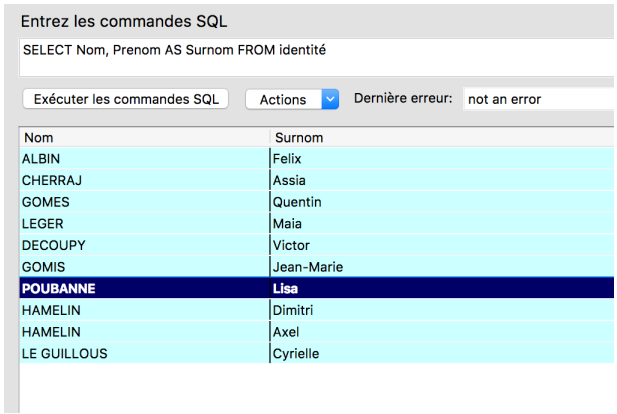
$$\sigma_{\text{année}=1998}(\text{identité}) \setminus \sigma_{\text{homme}=0}(\text{identité})$$

I.6 Renommage

Le renommage est l'opération algébrique consistant à donner un autre nom à une relation ou à un attribut.

SQL

```
SELECT Nom, Prenom AS Surnom
FROM identité;
```



Cette requête s'écrit en algèbre relationnelle :

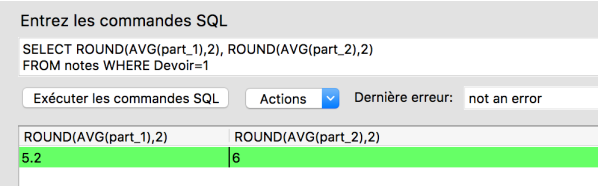
$$\rho_{\text{Prenom}\to\text{Surnom}}\left(\pi_{\text{Nom,Prenom}}(\text{identité})\right)$$

I.7 Fonctions d'agrégation

Les fonctions d'agrégation permettent de faire des calculs sur un groupe d'entrées sélectionnées. Par exemple, si on veut calculer la moyenne sur la partie 1 et 2 du premier devoir :

SQL

```
SELECT ROUND(AVG(part_1),2), ROUND(AVG(part_2),2)
FROM notes
WHERE Devoir=1;
```



Voici quelques fonction d'agrégation usuelle :

→ AVG calcule la moyenne ;

→ COUNT calcule le nombre de tuple sélectionnés

→ MAX et MIN pour le maximum et minimum

→ SUM pour le calcul de la somme

Certaines fonctions dont la valeur de retour est booléenne et qui servent à des critères de sélections d'autres requêtes : IN, ALL, ANY (ou SOME), EXISTS, NOT EXISTS. Elles apparaissent dans des sous-requêtes que nous verrons plus loin.

La fonction ROUND sur un attribut permet d'arrondir un résultat numérique. Cette fonction permet soit d'arrondir sans utiliser de décimal pour retourner un nombre entier (c'est-à-dire : aucun chiffre après la virgule), ou de choisir le nombre de chiffre après la virgule.

I.8 Opérateurs

Les opérateurs +, /, - et * peuvent également être utilisés. Par exemple part_1+part_2 permet de sommer la colonne part_1 et la colonne part_2 tuple par tuple (ce qui permet d'obtenir la note globale de chaque devoir de chaque élève).

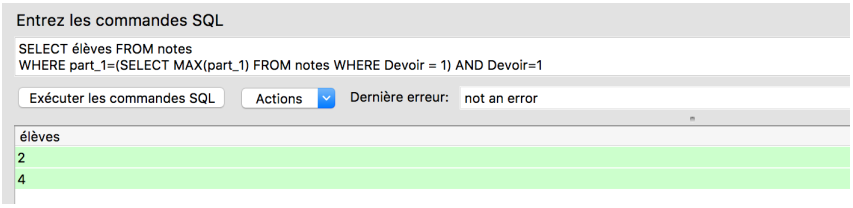
II Sous-requêtes

Il peut-être intéressant d'utiliser le résultat d'une requête R_1 à l'intérieur du critère d'une autre requête R_2 : on dit que la requête R_1 est une **sous-requête** de la requête R_2.

Par exemple, si on recherche dans la table *notes* les élèves ayant eu la meilleure note à la première partie du premier devoir :

SQL

```
SELECT élèves FROM notes
WHERE part_1=(SELECT MAX(part_1) FROM notes WHERE Devoir = 1) AND Devoir=1;
```

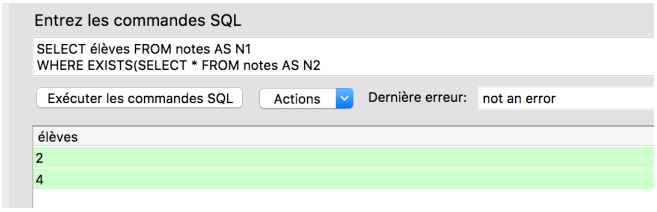


Utilisations du résultat d'une sous-requête avec ALL, EXISTS, ANY ou IN

On souhaite connaitre les élèves ayant eu la même note à la première partie du devoir 1 que l'élève numéro 2 :

SQL

```
SELECT élèves FROM notes AS N1
WHERE EXISTS(SELECT * FROM notes AS N2
WHERE N1.part_1=N2.part_1 AND N2.élèves=2 AND N1.Devoir=1 AND N2.Devoir=1);
```



En utilisant la fonction EXISTS, qui fournit le booléen Vrai si le résultat de la sous-requête est non vide et Faux sinon. Dans l'exemple précédent, on va donc chercher pour chaque élève de la table N1 si la table N2 est non vide, la table N2 est une table où on sélectionne que l'élève 2 et le devoir 1 et si la note de la partie 1 est la même que celle de l'élève de la table N1, elle est donc non vide que si l'élève de la table N1 a la même note que l'élève 2.

De même il existe la fonction *ALL* ou *ANY* qui signifie "vrai pour toutes les entrée" et "vrai pour au moins une des entrées" mais ce n'est pas le cas dans SQLite... Il faut passer par le négation *NOT* etc...

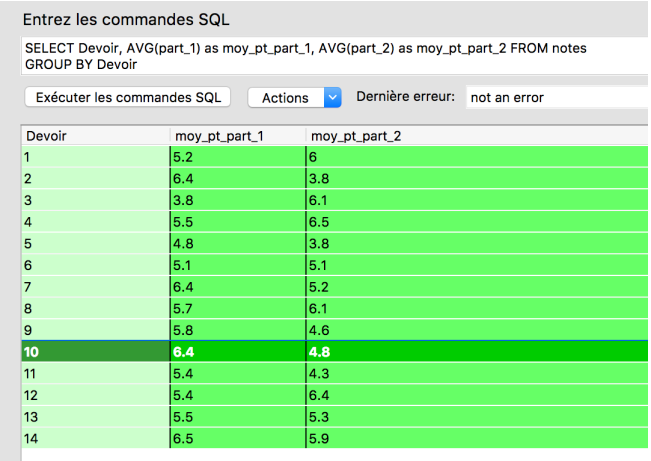
III Groupement de Tuples

III.1 GROUP BY

Cette fonction permet de partitionner les tuples renvoyés par une requête SELECT en différents groupes pour appliquer une fonction d'agrégation à chaque groupe.
Par exemple, si on veut calculer les moyennes de chaque partie de chaque devoir :

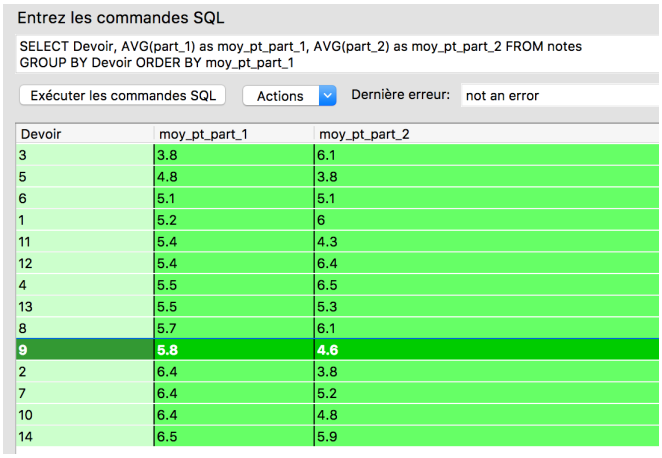
SQL

```
SELECT Devoir, AVG(part_1) as moy_pt_part_1, AVG(part_2) as moy_pt_part_2
FROM notes
GROUP BY Devoir;
```



Devoir	moy_pt_part_1	moy_pt_part_2
1	5.2	6
2	6.4	3.8
3	3.8	6.1
4	5.5	6.5
5	4.8	3.8
6	5.1	5.1
7	6.4	5.2
8	5.7	6.1
9	5.8	4.6
10	6.4	4.8
11	5.4	4.3
12	5.4	6.4
13	5.5	5.3
14	6.5	5.9

De plus, la fonction *ORDER BY* permet d'ordonner les lignes selon un critère :



Devoir	moy_pt_part_1	moy_pt_part_2
3	3.8	6.1
5	4.8	3.8
6	5.1	5.1
1	5.2	6
11	5.4	4.3
12	5.4	6.4
4	5.5	6.5
13	5.5	5.3
8	5.7	6.1
9	5.8	4.6
2	6.4	3.8
7	6.4	5.2
10	6.4	4.8
14	6.5	5.9

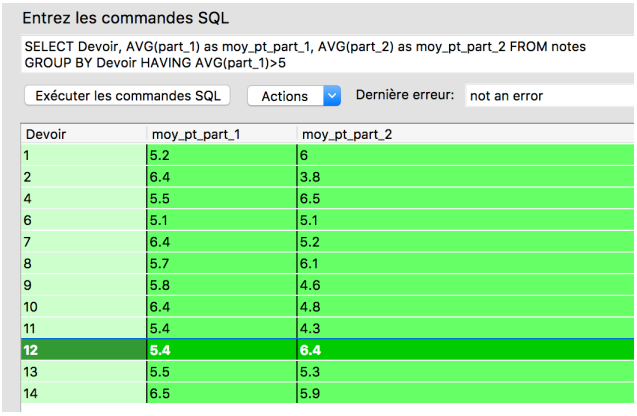
On rajoute DESC après l'attribut si on veut ordonner du plus grand au plus petit (par exemple ORDER BY moy_pt_part_1 DESC)

III.2 HAVING

Cette fonction effectue la même sélection que WHERE mais elle s'applique au groupe sélectionné via GROUP BY et non aux tuples eux-mêmes. Par exemple, si on veut garder dans l'exemple précédent les devoirs où la moyenne de la première partie dépasse 5 :

SQL

```sql
SELECT Devoir, AVG(part_1) as moy_pt_part_1, AVG(part_2) as moy_pt_part_2
FROM notes
GROUP BY Devoir
HAVING AVG(part_1)>5;
```



Devoir	moy_pt_part_1	moy_pt_part_2
1	5.2	6
2	6.4	3.8
4	5.5	6.5
6	5.1	5.1
7	6.4	5.2
8	5.7	6.1
9	5.8	4.6
10	6.4	4.8
11	5.4	4.3
12	5.4	6.4
13	5.5	5.3
14	6.5	5.9