Bases de données et SQL

Sommaire

I Introduction

I.1 Qu'est-ce qu'une base de données ?

Une base de données est un ensemble structuré d'informations, organisé de manière à pouvoir être consulté, mis à jour et partagé efficacement par plusieurs utilisateurs.

On utilise une base de données lorsqu'un simple tableau (par exemple dans un tableur) devient trop limité ou trop lourd à gérer. Les avantages principaux sont :

- rapidité d'accès aux données ;
- cohérence et fiabilité de l'information (moins d'erreurs de saisie) ;
- possibilité de gérer des volumes de données importants ;
- usage simultané par plusieurs utilisateurs.

I.2 Un premier exemple

On pourrait imaginer stocker les notes d'un groupe d'élèves dans un tableau unique :

Nom	Prenom	Devoir	type	coefficient	part_1	part_2
ALBIN	Felix	1	DS	2	1	4
CHERRAJ	Assia	1	DS	2	10	9
GOMES	Quentin	1	DS	2	3	8
LEGER	Maia	1	DS	2	10	5
DECOUPY	Victor	1	DS	2	4	6
GOMIS	Jean-Marie	1	DS	2	1	10
POUBANNE	Lisa	1	DS	2	9	3
HAMELIN	Dimitri	1	DS	2	8	4
HAMELIN	Axel	1	DS	2	5	7
LE GUILLOUS	Cyrielle	1	DS	2	1	4
ALBIN	Felix	2	DM	1	7	3
CHERRAJ	Assia	2	DM	1	9	3
GOMES	Quentin	2	DM	1	9	2
LEGER	Maia	2	DM	1	2	4
DECOUPY	Victor	2	DM	1	4	4
GOMIS	Jean-Marie	2	DM	1	8	1
POUBANNE	Lisa	2	DM	1	4	2
HAMELIN	Dimitri	2	DM	1	6	7
HAMELIN	Axel	2	DM	1	7	5
LE GUILLOUS	Cyrielle	2	DM	1	8	7
ALBIN	Felix	3	Interro	1	5	4
CHERRAJ	Assia	3	Interro	1	9	6
GOMES	Quentin	3	Interro	1	2	3
LEGER	Maia	3	Interro	1	2	8
DECOUPY	Victor	3	Interro	1	2	8

Tableau de données

Ce tableau pose plusieurs problèmes :

- Les noms sont répétés à chaque ligne, avec des risques de fautes ("Felix" sans accent, "Félix" avec accent, etc.).
- Les intitulés des devoirs varient ("DM", "Devoir maison", "interro", "interrogation"), ce qui complique le tri et les recherches.
- Retrouver toutes les notes d'un élève devient vite fastidieux si le tableau contient des milliers de lignes.

Pour de petites quantités de données, ces limites restent gérables à la main. Mais dès que le volume augmente, on a besoin d'un système plus puissant : la base de données relationnelle.

I.3 Motivation par un scénario réel

Imaginons un enquêteur de police cherchant un véhicule. Le témoin affirme qu'il s'agit d'une Renault rouge, coupé, immatriculée avec une plaque commençant par la lettre « A » et vendue par la concession « Bony Automobiles ».

Rechercher manuellement parmi des milliers de fiches serait impossible. Une base de données permet en revanche de retrouver instantanément tous les véhicules correspondant à ces critères.

```sql
SELECT *
FROM Voitures
WHERE Marque = 'Renault'
  AND Couleur = 'Rouge'
  AND Type = 'Coupé'
  AND Immatriculation LIKE 'A%'
  AND Concession = 'Bony Automobiles';
```

Cette requête renverra en une fraction de seconde tous les résultats pertinents.

II Organisation d'une base relationnelle

II.1 Principe

Une base de données relationnelle est constituée de plusieurs tables. Chaque table regroupe des informations sur un type d'objet (par exemple, les élèves, les devoirs, les notes).

Les tables sont reliées entre elles par des clés (identifiants uniques), ce qui permet d'éviter les répétitions et de garantir la cohérence des données.

II.2 Exemple : notes d'élèves

On peut décomposer le tableau initial en quatre tables :

- **Eleves** (Num,Nom,Prenom,Annee,sexe) (Num pour numéro)
- **Devoirs** (Num, Type, Coefficient)
- **Notes** (Num, Eleve, Devoir, Part_1, Part_2) (Part_1 pour la partie 1 du devoir etc.)
- **Type** (Num, Description)

Ainsi, le nom d'un élève ou le type d'un devoir n'apparaissent qu'une seule fois. Les erreurs sont réduites et les recherches sont beaucoup plus simples.

Num	Eleve	Devoir	Part_1	Part_2
1	1	1	1	4
2	**1**	**2**	**7**	**3**
3	1	3	5	4
4	1	4	4	9
5	1	5	9	3
6	1	6	3	8
7	1	7	6	6
8	1	8	8	5
9	1	9	9	4
10	1	10	9	7
11	1	11	3	3
12	1	12	4	9
13	1	13	7	9
14	1	14	6	8
15	2	1	10	9
16	2	2	9	3
17	2	3	9	6
18	2	4	3	9
19	2	5	8	1
20	2	6	6	1
21	2	7	7	5
22	2	8	6	10
23	2	9	2	7
24	2	10	1	8
25	2	11	1	9
26	2	12	9	9
27	2	13	5	7
28	2	14	5	8
29	3	1	3	8
30	3	2	9	2
31	3	3	2	3
32	3	4	4	2
33	3	5	6	2
34	3	6	4	5

Notes

Num	Nom	Prenom	Annee	Sexe
1	ALBIN	Felix	1998	1
2	CHERRAJ	Assia	1998	0
3	GOMES	Quentin	1997	1
4	**LEGER**	**Maia**	**1998**	**0**
5	DECOUPY	Victor	1997	1
6	GOMIS	Jean-Marie	1997	1
7	POUBANNE	Lisa	1998	0
8	HAMELIN	Dimitri	1998	1
9	HAMELIN	Axel	1998	1
10	LE GUILLOUS	Cyrielle	1998	0

Eleves

Num	Type	Coefficient
1	1	2
2	**2**	**1**
3	3	1
4	1	1
5	2	2
6	1	3
7	2	1
8	3	2
9	1	1
10	2	2
11	1	3
12	2	1
13	1	1
14	2	1
15	1	2
16	1	3

Devoir

Num	Description
1	DS
2	**DM**
3	Interro

Type

III Langage SQL

III.1 Qu'est-ce que SQL ?

Le **SQL** (*Structured Query Language*) est le langage standard permettant de dialoguer avec une base de données relationnelle.

Il comporte plusieurs volets :

- **DDL** (Data Definition Language) : créer, modifier, supprimer des tables.
- **DML** (Data Manipulation Language) : insérer, mettre à jour, supprimer des données.
- **DQL** (Data Query Language) : interroger les données avec `SELECT`.

III.2 Vocabulaire

- **tuple :** les lignes sont les **entrées** ou **tuples** ou encore **entités**. Ces tuples ne sont pas nommés. Exemple : (1,ALBIN,Felix,1998,1).

- **attribut :** les colonnes sont appelées les **champs**, les **attributs** ou encore les **propriétés** : dans l'exemple précédent, les attributs de la table *eleve* sont *Num* (numéro de l'élève), *Nom*, *Prenom*, *Annee*, *Sexe*.

- **table :** tableau contenant les données. C'est un sous-ensemble du produit cartésien de plusieurs attributs. Par exemple, la table identité contient les éléments (1,ALBIN,Felix,1998,1), (2,CHERRAJ,Assia,1998,0), etc... On l'appelle également relation (langage algébrique) ou type entité.

- **domaine :** sous SQL, un domaine est décrit par un type (INT,FLOAT,etc...), auquel on peut ajouter des contraintes (tous différents, positif, etc...). Les types les plus usuels sont :

  → Le type VARCHAR (ou CHAR VARYING) permet de coder des chaînes de longueur variables, mais la longueur maximale est fixée. Par exemple VARCHAR(20) désigne le type des chaines alphanumériques formées d'au plus 20 caractères.

  → CHAR désigne le type des chaînes alphanumériques de longueur fixe, par exemple CHAR(10) désigne le type des chaînes alphanumériques formées de 10 caractères.

  → NUMERIC (ou DECIMAL ou DEC) qui permet de coder des nombres décimaux de façon exacte.

  → Le type FLOAT qui permet de coder des décimaux en base 2 de façon approchée (mais les calculs seront plus rapides qu'avec le type NUMERIC).

  → INT (ou INTEGER) qui permet de coder des entiers.

  → TEXT qui permet de coder des textes (éventuellement longs) de longueur indéterminée.

  → DATE et TIME pour la date et l'heure.

  Il faut se souvenir que chaque attribut possède un type qui lui est propre (éventuellement son domaine peut être restreint par des contraintes). Par exemple, dans la relation *Eleve*, l'attribut *Sexe* est un booléen (1 si c'est un homme, 0 si c'est une femme).

  Remarque 1.1. *Les attributs ne sont pas ordonnés, par contre ils possèdent tous un nom et peuvent être appelées par celui-ci. Les lignes ne sont pas ordonnées également mais elles doivent être toutes différentes.*

- **clé primaire :** Une clé primaire permet d'identifier de manière univoque chaque tuple d'une relation. C'est un sous-ensemble des attributs de la relation qui permet de distinguer chaque entrée de la relation, mais telle que, si on retire n'importe lequel des attributs de ce sous-ensemble, alors on ne peut plus distinguer certaines entrées de la relation. Par exemple dans la relation identité de notre exemple, à laquelle on aurait enlevé l'attribut *Num*, on aurait pu choisir le sous-ensemble constitué des attributs *Nom* et *Prenom*, mais deux personnes peuvent avoir le même nom dans certains cas. Le sous-ensemble contenant que *Num* convient dans tous les cas. En pratique la clé est souvent limitée à un seul attribut, qui prend des valeurs distinctes pour chaque entrée de la relation. Cet attribut sera identifié par le terme Cle ou Identifiant ou Id (éventuellement suivi de l'objet à identifier : Villes, Nom, Étudiant, Professeur.)

- **clé étrangère :** Une clé étrangère permet de référencer les entrées de certaines relations avec la clé (primaire) d'une autre relation : dans notre exemple, l'attribut *Eleve* de la relation *Notes* est une clé étrangère qui référence la clé primaire de la relation *Eleves*.

- **schéma relationnel :** Le schéma d'une relation est la donnée du nom de la relation et de chacun de ses attributs (avec ou sans mention de leurs domaines). Dans nos exemples, nous avons les deux schémas : *Eleves(Num,Nom,Prénom,Année,Sexe)*. Notez que l'on utilisera un symbole pour distinguer les clés primaires…
- **schéma de base de données** : est la donnée des différentes relations de la base, chacune avec ses attributs, et des références entre clés étrangères et clés primaires des différentes relations d'une base de données.

Exercice 1.2

Effectuer le schéma de base de données de l'exemple de ce cours :

E

Définition 1.3: entité

On appelle entité un objet (concret ou abstrait) qui peut être reconnue distinctement et qui est caractérisée par son unicité.

Exemple 1.4

un élève (Albin félix, Poubanne Lisa, etc..), la copie d'un élève à un devoir (la copie de Lisa au devoir 3), etc ...

Définition 1.5: type entité

Un type entité désigne un ensemble d'entités qui possèdent une sémantique et des propriétés communes.

Exemple 1.6

des élèves, des copies de devoir (plus précisément, les notes de ces copies) , des notes d'interrogation, etc..

Définition 1.7: association

On appelle **association** un lien entre plusieurs entités

Exemple 1.8

L'**obtention** de la note de 5/20 par Albin Félix au premier devoir

Définition 1.9: type association

Un type association désigne un ensemble d'associations qui possèdent les mêmes caractéristiques. Le type association décrit un lien entre plusieurs types entités. Les associations de ce type association lient les entités des tables.

Exemple 1.10

L'obtention des notes aux devoirs entre le type entité **élèves** et le type entité **notes de devoirs**.

Définition 1.11: cardinalité

On appelle **cardinalité** d'un lien reliant un type association à un type entité le nombre de fois minimal et maximal d'interventions d'une entité du type entité dans une association du type association.

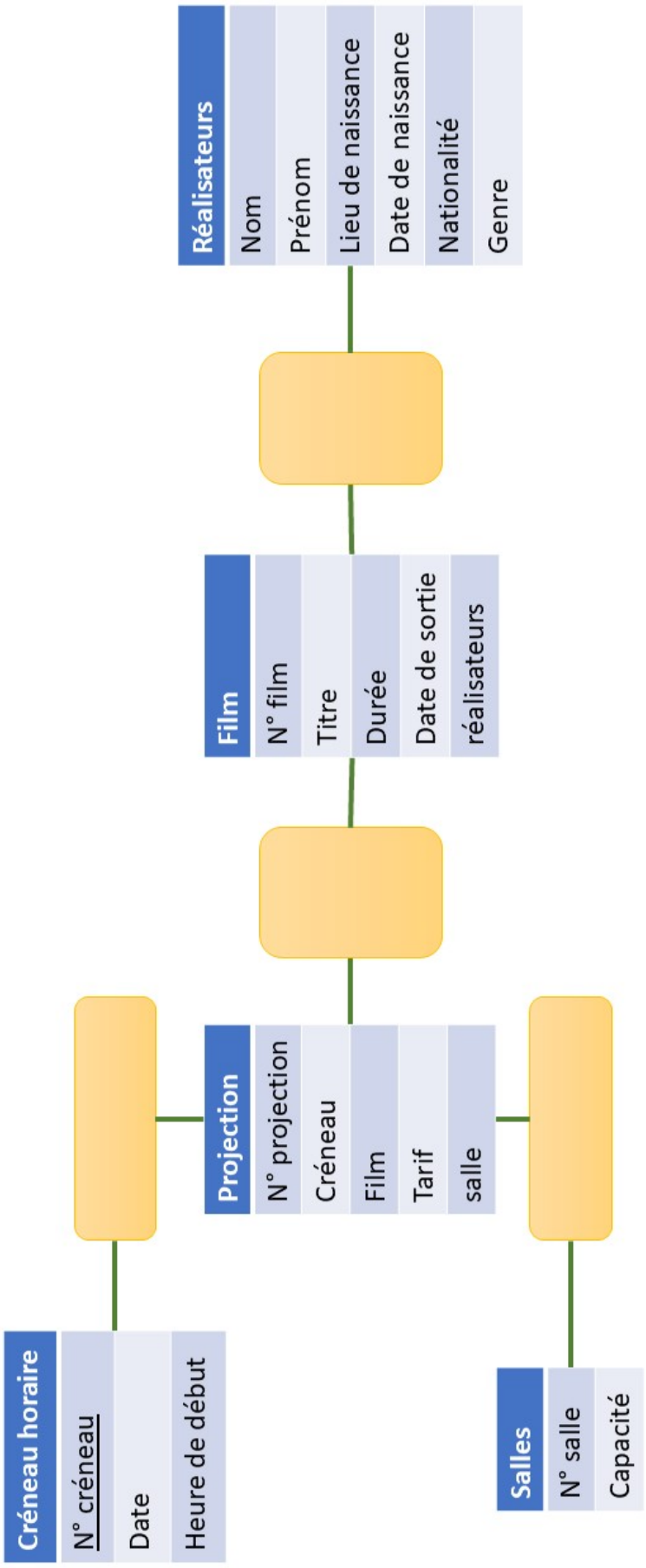
Remarque 1.12. *Même si on connait le nombre maximal de fois qu'on utilise une entité (par exemple ici, si on sait qu'il y a 12 devoirs, on utilise donc au plus 12 fois un élève) on ne marquera pas ce nombre mais la variable n. En effet, comme les tables ne sont pas figés, ce nombre pourrait changer dans le temps (par exemple si on effectue un* 13^e *devoir, alors on devrait passer de 12 à 13).*

Remarque 1.13. *La cardinalité minimale admise est soit 0 (on peut ne pas utiliser toutes les entités) soit* 1 *(on doit toutes les utiliser au moins une fois). La cardinalité maximale admise est soit* 1 *(on l'utilise au plus une fois), soit n (on l'utilise autant de fois qu'on veut). Il y a donc* 4 *possibilités :*

- *0,1 : une entité peut être utilisée au plus qu'une seule fois et peut ne pas être utilisé.*
- *0,n : elle peut être utilisée sans limitation.*
- *1,1 : elle peut être utilisée une et une seule fois.*
- *1,n : elle doit être utilisée au moins une fois et sans limitation.*

Exemple 1.14

Le type association **obtention des notes aux devoirs** liant le type entité **élèves** au type entité **notes aux devoirs** et de cardinalité 0,n pour les élèves, 1,1 pour les notes de devoirs.


Réalisateurs
Nom
Prénom
Lieu de naissance
Date de naissance
Nationalité
Genre
Film
N° film
Titre
Durée
Date de sortie
réalisateurs
Projection
N° projection
Créneau
Film
Tarif
salle
Créneau horaire
N° créneau
Date
Heure de début
Salles
N° salle
Capacité

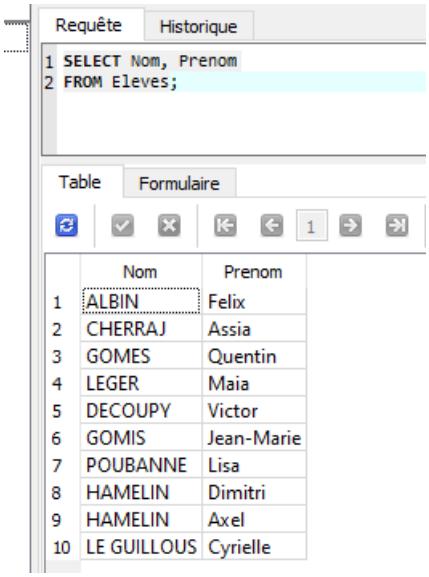
IV Requêtes SQL

IV.1 Premières requêtes simples

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 *Eleves* en ne gardant que les attributs *Nom* et *Prénom*, la requête SQL est :

```
SELECT Nom, Prenom
FROM Eleves;
```



Remarque 1.15. *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,

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

ne donnera que les 4 premières lignes du résultat précédent.

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

```
SELECT Nom, Prenom
FROM Eleves
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.

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

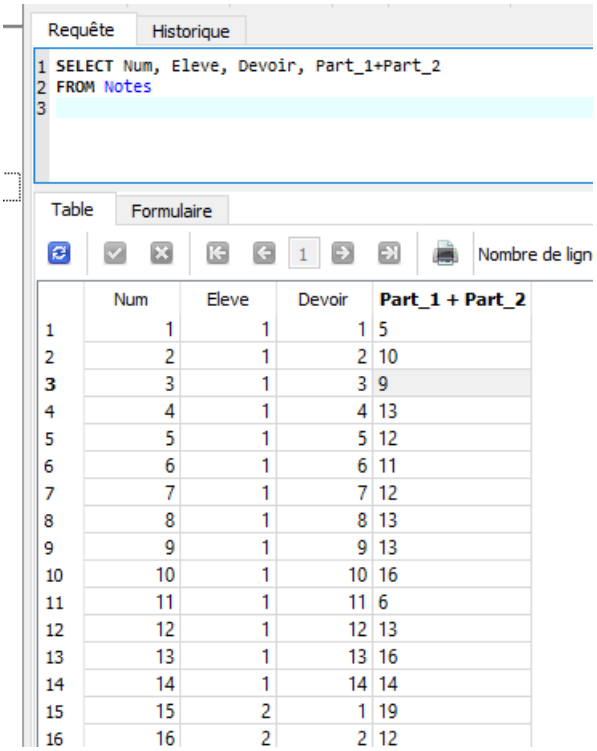
Les lignes seront ordonnées selon les noms des élèves par ordre ascendant (plus petit au plus grand). Si on veut par ordre descendant (plus grand au plus petit) :

```
SELECT Nom, Prenom
FROM Eleves
ORDER BY Nom DESC;
```

Opérations lors d'une projection

On peut effectuer certaines opérations sur les attributs, notamment toutes les opérations algébriques élémentaires : $+$, $-$, $*$, $/$. Par exemple, si on veut les notes de chaque devoir et non la note sur chaque partie :

```
SELECT Num, Eleve, Devoir, Part_1+Part_2
FROM Notes
```

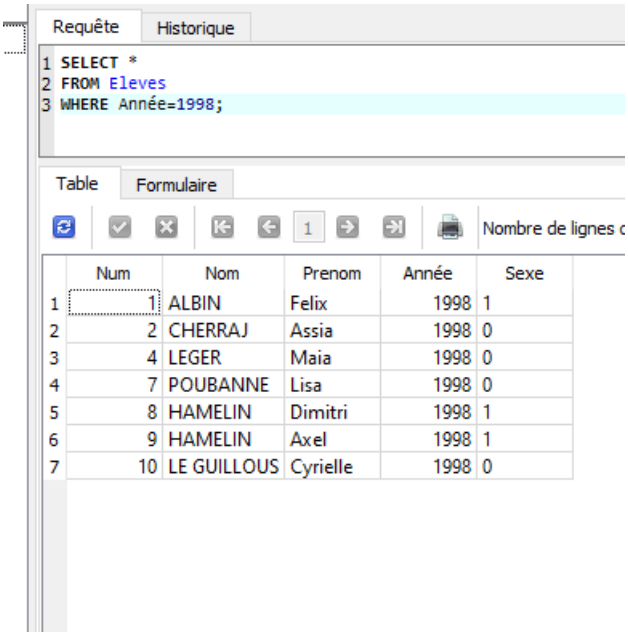


	Num	Eleve	Devoir	Part_1 + Part_2
1	1	1	1	5
2	2	1	2	10
3	3	1	3	9
4	4	1	4	13
5	5	1	5	12
6	6	1	6	11
7	7	1	7	12
8	8	1	8	13
9	9	1	9	13
10	10	1	10	16
11	11	1	11	6
12	12	1	12	13
13	13	1	13	16
14	14	1	14	14
15	15	2	1	19
16	16	2	2	12

Sélection

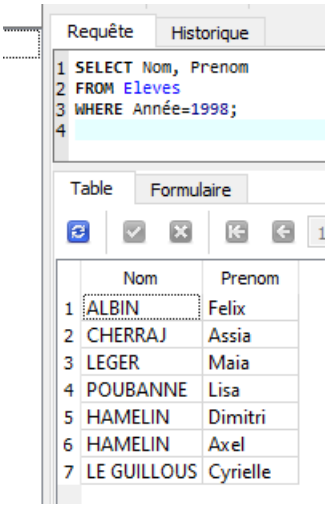
La **sélection** est l'opération permettant de ne choisir que les entrées (donc les lignes) d'une relation vérifiant une certaine condition. Par exemple si l'on cherche les élèves nés en 1998, la requête SQL est :

```
SELECT *
FROM Eleves
WHERE Année=1998;
```



Remarque 1.16. *L'étoile permet de garder toutes les colonnes sans les rappeler, mais on peut combiner projection et sélection :*

```
SELECT Nom, Prenom
FROM Eleves
WHERE Année=1998;
```

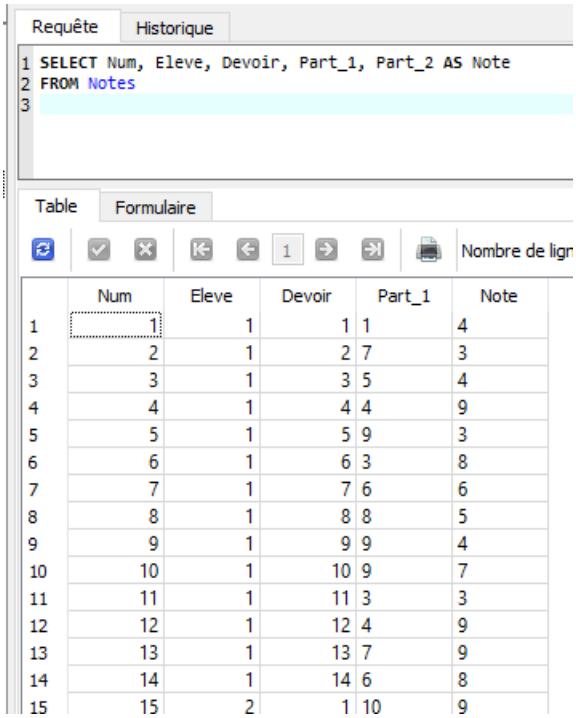


Remarque 1.17. *Les opérateurs logiques possibles (à connaître) sont =, <> (≠), <, <=, >, >=, AND, OR, NOT.*

Renommage

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

```
SELECT Num, Eleve, Devoir, Part_1+Part_2 AS Note
FROM Notes
```



	Num	Eleve	Devoir	Part_1	Note
1	1	1	1	1	4
2	2	1	2	7	3
3	3	1	3	5	4
4	4	1	4	4	9
5	5	1	5	9	3
6	6	1	6	3	8
7	7	1	7	6	6
8	8	1	8	8	5
9	9	1	9	9	4
10	10	1	10	9	7
11	11	1	11	3	3
12	12	1	12	4	9
13	13	1	13	7	9
14	14	1	14	6	8
15	15	2	1	10	9

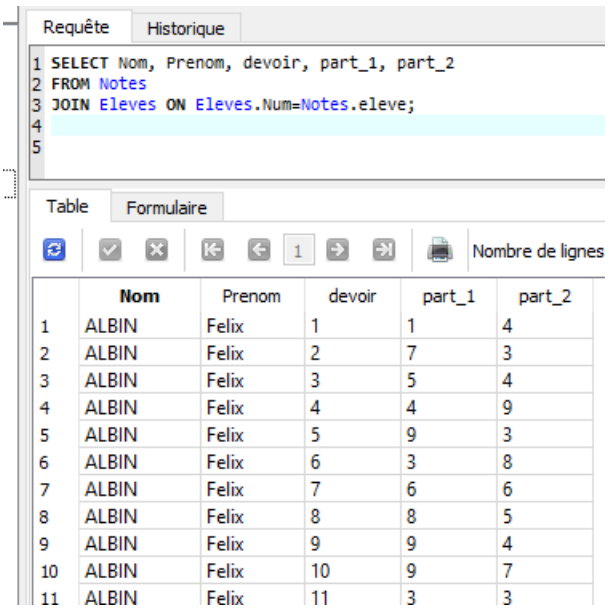
IV.2 Jointure entre tables, autojointure

Une jointure en SQL est une opération qui permet de relier plusieurs tables entre elles pour obtenir une seule table de résultats. Elle repose sur une condition de correspondance, souvent une égalité entre des clés (par exemple une clé primaire et une clé étrangère).

Par exemple, si on veut retrouver le nom et le prénom des élèves avec les notes :

```
SELECT Nom, Prenom, devoir, part_1, part_2
FROM Notes
JOIN Eleves ON Eleves.Num=Notes.eleve;
```

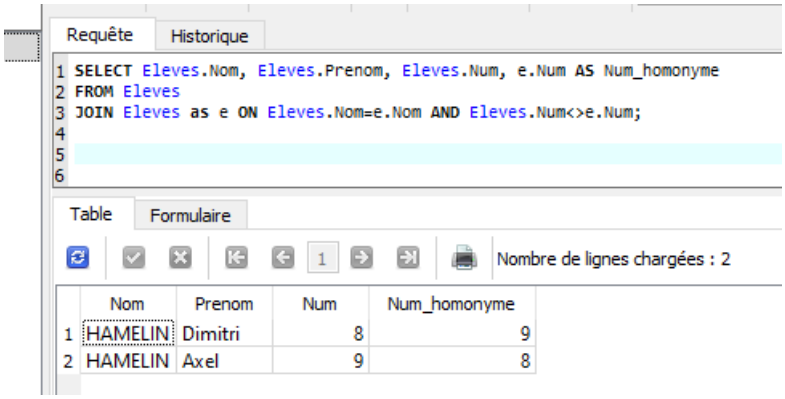
ce qui donne :



	Nom	Prenom	devoir	part_1	part_2
1	ALBIN	Felix	1	1	4
2	ALBIN	Felix	2	7	3
3	ALBIN	Felix	3	5	4
4	ALBIN	Felix	4	4	9
5	ALBIN	Felix	5	9	3
6	ALBIN	Felix	6	3	8
7	ALBIN	Felix	7	6	6
8	ALBIN	Felix	8	8	5
9	ALBIN	Felix	9	9	4
10	ALBIN	Felix	10	9	7
11	ALBIN	Felix	11	3	3

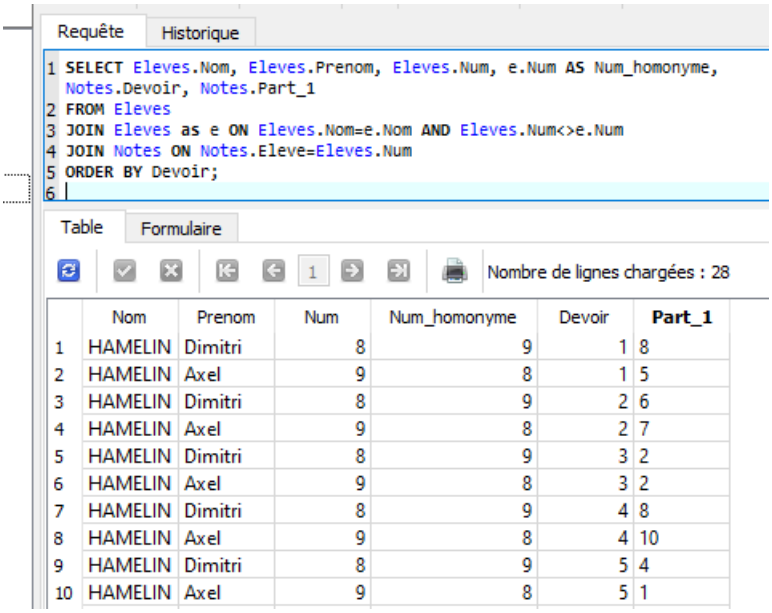
On peut effectuer des **autojointures**, c'est-à-dire la jointure d'une table avec elle-même, il est alors utile de **renommer** une table. De plus, en cas d'ambiguïté, on peut écrire *table.attribut* plutôt que juste le nom de l'attribut. Par exemple, si on veut vérifier qu'il y a des élèves qui ont le même nom :

```
SELECT Eleves.Nom, Eleves.Prenom, Eleves.Num, e.Num AS Num_homonyme
FROM Eleves
JOIN Eleves as e ON Eleves.Nom=e.Nom AND Eleves.Num<>e.Num
```



Remarque 1.18. *Il est bien sûr possible de joindre plusieurs tables, comme dans l'exemple suivant où on compare les notes des deux homonymes :*

```
SELECT Eleves.Nom, Eleves.Prenom, Eleves.Num, e.Num AS Num_homonyme,
 Notes.Devoir, Notes.Part_1
FROM Eleves
JOIN Eleves as e ON Eleves.Nom=e.Nom AND Eleves.Num<>e.Num
JOIN Notes ON Notes.Eleve=Eleves.Num
ORDER BY Devoir;
```

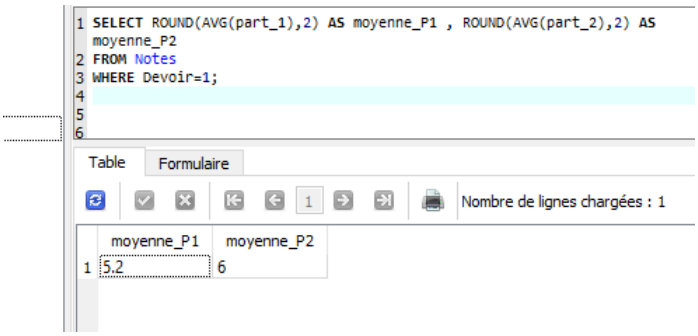


IV.3 Agrégations

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 :

```
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

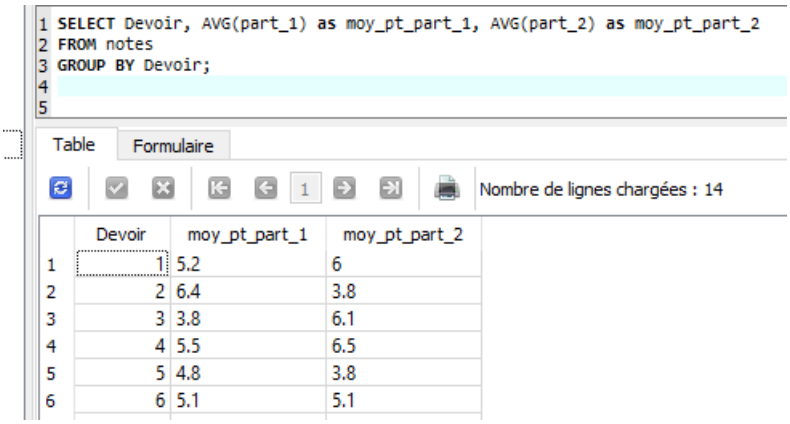
→ SUM pour le calcul de la somme

→ ROUND permet d'arrondir après la virgule avec le nombre de décimales demandé (cf exemple)

GROUP BY

Cette fonction permet de partitionner les tuples renvoyés par une requête 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 :

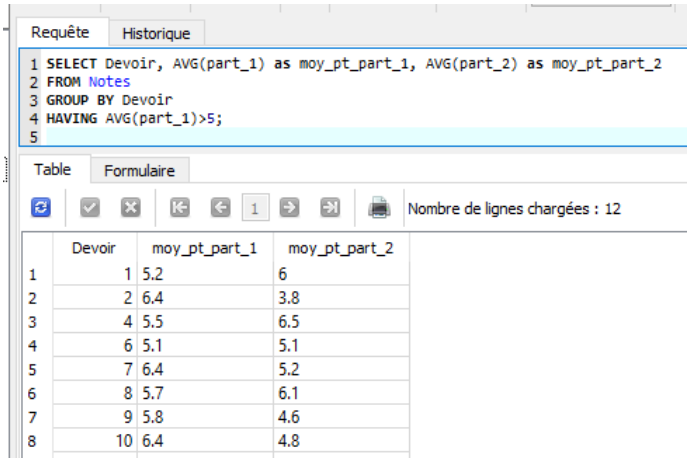
```
SELECT Devoir, AVG(part_1) as moy_pt_part_1, AVG(part_2) as moy_pt_part_2
FROM notes
GROUP BY Devoir;
```



Filtrage des agrégats avec 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 :

```
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;
```



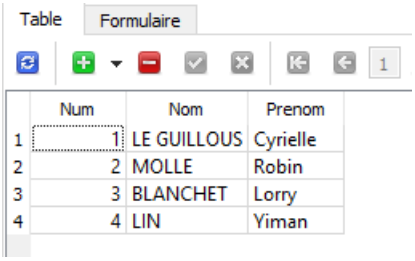
	Devoir	moy_pt_part_1	moy_pt_part_2
1	1	5.2	6
2	2	6.4	3.8
3	4	5.5	6.5
4	6	5.1	5.1
5	7	6.4	5.2
6	8	5.7	6.1
7	9	5.8	4.6
8	10	6.4	4.8

Il est important de comprendre que :

- WHERE filtre les lignes avant l'agrégation.
- HAVING filtre les groupes après l'agrégation (après un GROUP BY).

IV.4 Opérateurs ensemblistes

Dans cette section, on a rajouté une table nommé Eleves2 que voici :



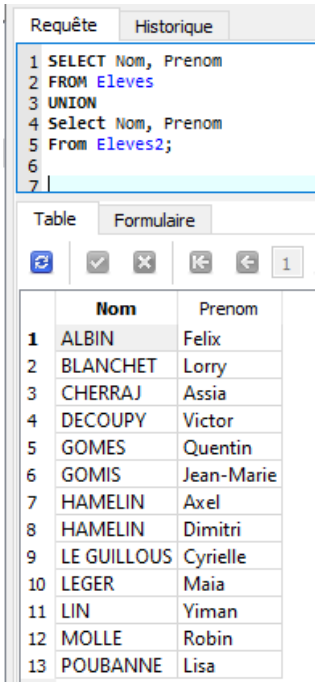
	Num	Nom	Prenom
1	1	LE GUILLOUS	Cyrielle
2	2	MOLLE	Robin
3	3	BLANCHET	Lorry
4	4	LIN	Yiman

Union

L'union est une opération ensembliste que l'on peut réaliser sur des relations/tables à condition qu'elles aient la même structure, c'est-à-dire le même nombre d'attribut. Par exemple, si je veux trouver tous les noms des élèves apparaissant dans l'une des deux tables :

```
SELECT Nom, Prenom
FROM Eleve
UNION
Select Nom, Prenom
From Eleve2;
```

ce qui donne :

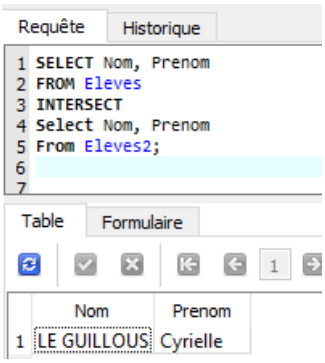


Intersection

On peut également réaliser des intersections entre relations/tables ayant la même structure. Par exemple, si on cherche le (Nom,Prenom) des élèves apparaissant dans les deux tables :

```
SELECT Nom, Prenom
FROM Eleves
INTERSECT
Select Nom, Prenom
From Eleves2;
```

ce qui donne :

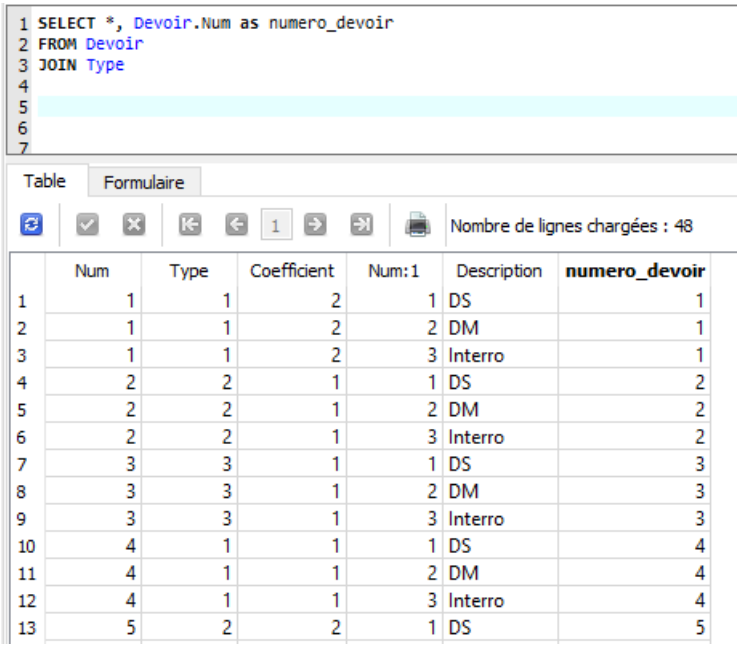


Produit cartésien

Il a le même principe qu'en algèbre, c'est-à-dire que si vous avez un ensemble A de tuples $(a_1, ..., a_n)$ et un ensemble B de tuples $(b_1, ..., b_m)$, le produit cartésien de ces deux ensemble donne l'ensemble des tuples $(a_1, ..., a_n, b_1, ..., b_m)$ avec $(a_1, ..., a_n)$ dans A et $(b_1, ..., b_m)$ dans B. Voici un exemple :

```sql
SELECT *, Devoir.Num as numero_devoir
FROM Devoir
JOIN Type;
```

ce qui donne :



```
1 SELECT *, Devoir.Num as numero_devoir
2 FROM Devoir
3 JOIN Type
4
5
6
7
```

Table | Formulaire

Nombre de lignes chargées : 48

	Num	Type	Coefficient	Num:1	Description	numero_devoir
1	1	1	2	1	DS	1
2	1	1	2	2	DM	1
3	1	1	2	3	Interro	1
4	2	2	1	1	DS	2
5	2	2	1	2	DM	2
6	2	2	1	3	Interro	2
7	3	3	1	1	DS	3
8	3	3	1	2	DM	3
9	3	3	1	3	Interro	3
10	4	1	1	1	DS	4
11	4	1	1	2	DM	4
12	4	1	1	3	Interro	4
13	5	2	2	1	DS	5

Remarque 1.19. *Le produit cartésien n'a pas vraiment d'intérêt ainsi. Dans le cas précédent, on remarque que cela aurait un intérêt de sélectionner que les triplets ou l'attribut type représente bien le type de devoir du devoir concerné. Autrement dit, il faut sélectionner les triplets où le numéro dans Type correspond au numéro indiqué dans l'attribut type dans Devoir. Le devoir numéro 1 a pour valeur dans Devoir.type : 1, c'est donc un DS, il faut donc garder le triplet (1,2,DS,1) et supprimer tous les triplets de la forme (a,b,c,1), c'est le principe de la jointure.*

Différence ensembliste

Dans le même principe que l'union et l'intersection, EXCEPT permet de faire des différences ensemblistes.

```sql
SELECT attribut1, attribut2
FROM Table1
EXCEPT
SELECT attribut1, attribut2
FROM Table2;
```

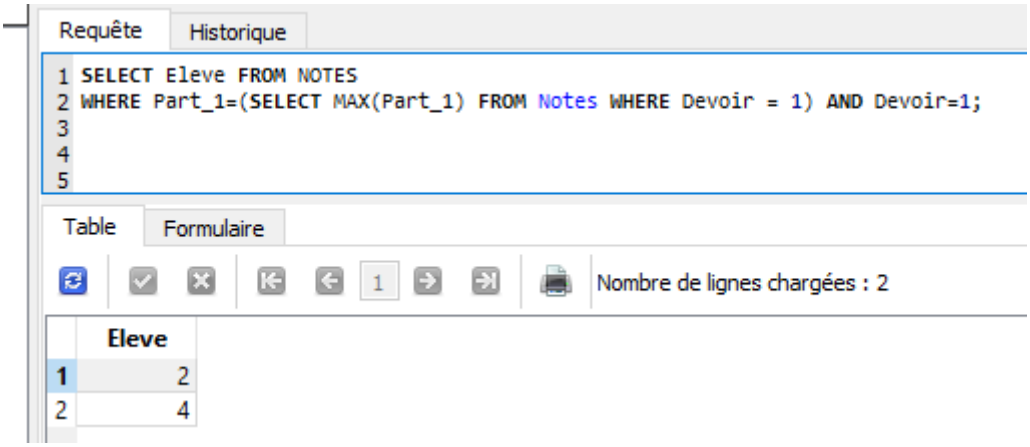
permet de voir les couples (attribut1,attribut2) qui sont dans Table1 mais pas dans Table2.

IV.5 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 :

```
SELECT Eleve FROM NOTES
WHERE Part_1=(SELECT MAX(Part_1) FROM Notes WHERE Devoir = 1) AND Devoir=1;
```



IV.6 Syntaxe générale

Les différentes fonctions doivent être écrites dans un ordre précis, voici un exemple avec la majeure partie des fonctions au programme.

```
SELECT Eleves.Année, AVG(Part_1+Part_2) as Moy
FROM Notes
JOIN Eleves
ON Eleves.Num=Notes.Eleve
WHERE Devoir>=6
GROUP BY Eleves.Année
HAVING Moy>10
ORDER BY Moy DESC
LIMIT 1
```

Exercice 1.20: A quoi sert la requête précédente ?

E