Bases de données

I. Introduction

Une base de données est un ensemble organisé d'informations, destiné à être stocké, consulté et manipulé efficacement. Nous utilisons tous les jours ce type de système, souvent sans même nous en rendre compte. Lorsque vous réservez un billet de train, que vous consultez le solde de votre compte en ligne ou que vous cherchez un livre dans le catalogue de la bibliothèque, vous interrogez en réalité une base de données.

Les bases de données sont ainsi omniprésentes :

- ⋆ les *banques* s'en servent pour gérer les comptes et les transactions ;
- ⋆ les sites de *commerce en ligne* enregistrent les produits, les clients et les commandes ;
- ⋆ les *réseaux sociaux* stockent les profils, messages et relations entre utilisateurs ;
- ⋆ les *établissements scolaires* gèrent leurs élèves, cours et notes dans des bases de données.

Définition 1 : *Système de Gestion de Bases de Données*

Un système de gestion de bases de données (SGBD) est un logiciel permettant de créer, organiser et manipuler des bases de données. Il fournit des outils pour stocker les informations, garantir leur cohérence, gérer l'accès concurrent de plusieurs utilisateurs et interroger les données efficacement.

Dans ce chapitre, nous allons étudier deux points essentiels :

- ⋆ comment sont organisées les bases de données relationnelles : c'est le modèle relationnel ;
- ⋆ comment interroger une base de données relationnelle pour en extraire des informations : c'est le rôle du langage SQL (*Structured Query Language*).

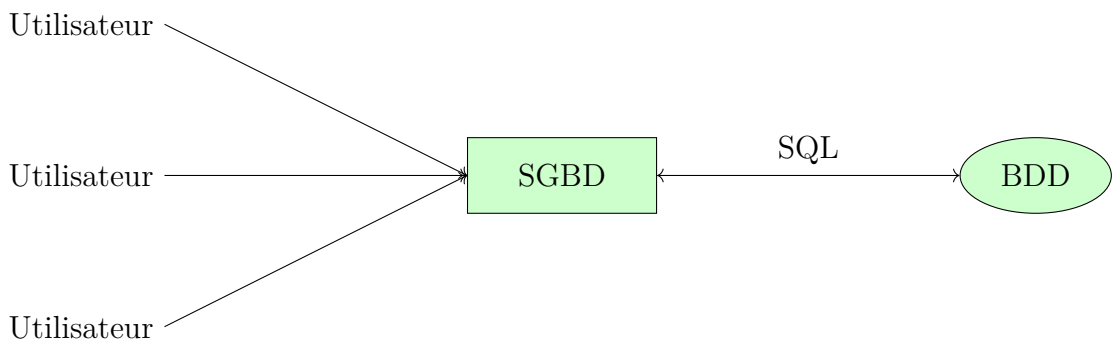

FIGURE 1 – Communication avec une base de données

II. Organisation des bases de données relationnelles

1) Modèle relationnel : entités et associations

Le modèle relationnel organise les données sous forme d'entités décrites par des attributs et de relations entre entités appelées associations.

Définition 2 : *Entités et associations*

Une entité modélise un objet du domaine (par exemple un *élève*, une *classe*, un *professeur*). Une association modélise un lien sémantique entre entités (par exemple *enseigner* lie un professeur et une classe pour une *matière*).

Définition 3 : *Cardinalité d'une association*

La cardinalité d'une association indique combien d'occurrences d'une entité peuvent être liées à une occurrence de l'autre. On distingue trois types de cardinalités :

- $1-1$: un élément de l'entité E_1 est associé à **au plus un** élément de l'entité E_2 et réciproquement ;
- $1-*$: un élément de l'entité E_1 peut être associé à **plusieurs** éléments de l'entité E_2 mais un élément de l'entité E_2 est associé à **au plus un** élément de l'entité E_1 ;
- $*-*$: un élément de l'entité E_1 peut être associé à **plusieurs** éléments de l'entité E_2 et réciproquement.

Exemple : Prenons l'exemple d'un lycée. Nous pouvons identifier plusieurs entités :

- les élèves
- les professeurs
- les classes

À partir de ces entités, nous pouvons identifier plusieurs associations :

- « appartenir » : un élève appartient à une classe $(1-*)$
- « enseigner » : un professeur enseigne dans une classe $(*-*)$
- « évaluer » : une note est attribuée à un élève par un professeur (relation ternaire)

2) Vocabulaire fondamental

Pour comprendre et manipuler des bases de données relationnelles, il est nécessaire de maîtriser quelques notions de base.

Définition 4 : *Vocabulaire des bases de données relationnelles*

Une table (ou relation) est un tableau à deux dimensions composé de lignes et de colonnes.

- Chaque colonne ou attribut décrit une caractéristique des données.
- Chaque ligne ou enregistrement représente une donnée particulière.

Exemple : La table `Eleve` ci-dessous décrit des élèves par leur identifiant, nom, prénom, email et date de naissance :

id	nom	prenom	email	date_naissance
1	Martin	Alice	alice.martin@gmail.com	2008-05-07
2	Dubois	Lucas	lucas.dubois@orange.com	2007-05-31
3	Bernard	Emma	emma.bernard@laposte.com	2008-11-05

Définition 5 : *Domaine d'un attribut*

Le domaine d'un attribut est l'ensemble des valeurs possibles qu'il peut prendre. Par exemple, le domaine de l'attribut `date_naissance` est l'ensemble des dates valides, tandis que celui de `prenom` est l'ensemble des chaînes de caractères.

Remarque : La notion de domaine permet de **garantir l'intégrité des données**. Par exemple, si l'attribut `date_naissance` est défini comme une date, le SGBD empêchera l'insertion d'une valeur incohérente comme 2008−13−05.

Définition 6 : *Schéma d'une table*

Le schéma d'une table décrit sa structure : il précise son nom, la liste des attributs avec leur domaine.

Modélisation des entités

En pratique, chaque entité est représentée par une table de la base de données. Ainsi, dans l'exemple précédent, la base de données `lycee` contiendra trois tables : `Eleve`, `Professeur`, `Classe`.

Chacune d'entre elles peut être décrite par un certain nombre d'attributs :

```
Eleve(id: INT, nom: TEXT, prenom: TEXT, email: TEXT, date_naissance: DATE)

Professeur(id: INT, nom: TEXT, prenom: TEXT, matiere: TEXT)

Classe(id: INT, nom: TEXT)
```

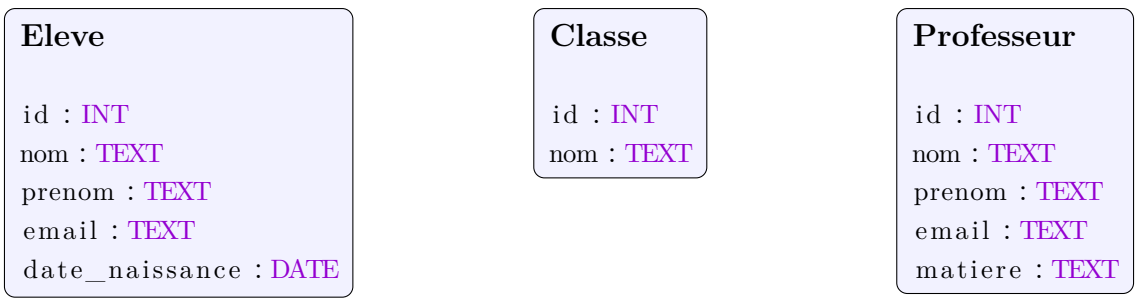

FIGURE 2 – Base de données `lycee`

3) Clés primaires et clés étrangères

Pour garantir l'unicité et les liens entre les données, on utilise des clés.

Définition 7 : *Clé primaire*

Une clé primaire est un ensemble d'attributs permettant d'identifier de manière unique chaque enregistrement d'une table.

Remarque :

- ⋆ Une table ne peut avoir qu'**une seule clé primaire**. En revanche, plusieurs choix peuvent être possibles. Dans le schéma d'une table, on **souligne** les attributs de la clé primaire pour indiquer celle que l'on a choisi.
- ⋆ Le SGBD impose que l'ensemble des valeurs des attributs de la clé primaire soit **unique** pour chaque ligne.
- ⋆ Dans la majorité des cas, la clé primaire est constituée d'un seul attribut appelé `id` pour identifiant.

Exemple : Dans la base de données `lycee`, l'attribut `id` peut être choisi comme clé primaire dans chacune des tables :

```
Eleve(id: INT, nom: TEXT, prenom: TEXT, email: TEXT, date_naissance: DATE)

Professeur(id: INT, nom: TEXT, prenom: TEXT, email: TEXT, matiere: TEXT)

Classe(id: INT, nom: TEXT)
```

En revanche, l'attribut `nom` ne peut pas être choisi comme clé primaire car deux élèves distincts peuvent avoir le même nom de famille. L'ensemble des deux attributs `nom` et `prenom` n'est pas non plus un bon choix pour une clé primaire car cela interdirait d'avoir des élèves homonymes.

Si on souhaite enregistrer dans la base les absences des élèves, on pourrait utiliser le schéma suivant :

```
Absence(id_eleve: INT, date: DATE, motif: TEXT, justifie: BOOLEAN)
```

En effet, l'identifiant de l'élève seul ne suffit pas car un élève peut avoir plusieurs absences. De même la date seule ne suffit pas car plusieurs élèves peuvent être absents le même jour.

Définition 8 : *Clé étrangère*

Une clé étrangère est un ensemble d'attributs d'une table qui font référence à la clé primaire d'une autre table. Elle permet de représenter une association entre deux entités.

Modélisation des associations

La représentation d'une association entre deux entités dans une base de données va dépendre de plusieurs facteurs, en particulier de la cardinalité de l'association :

- ⋆ Une association entre deux entités de cardinalité $1 - *$ sera représentée par un ou plusieurs attributs dans la deuxième table. Ainsi, l'association « appartenir (à une classe) » sera représentée par un attribut `id_classe` dans la table `Eleve`. Cet attribut faisant référence à la clé primaire de la table `Classe`, il s'agit donc d'une clé étrangère.
- ⋆ Une association entre deux entités de cardinalité $* - *$ sera représentée par une table dédiée à cette association. Ainsi, l'association « enseigner » sera représentée par une table `Enseignement` contenant un attribut `id_classe` et un attribut `id_professeur` (et éventuellement d'autres attributs comme la `matiere` par exemple). Ces deux attributs sont alors des clés étrangères dans la table `Enseignement`.

Remarque : Les clés primaires et étrangères permettent de **garantir l'intégrité référentielle** : le SGBD interdit, par exemple, d'affecter un élève à une classe qui n'existe pas.

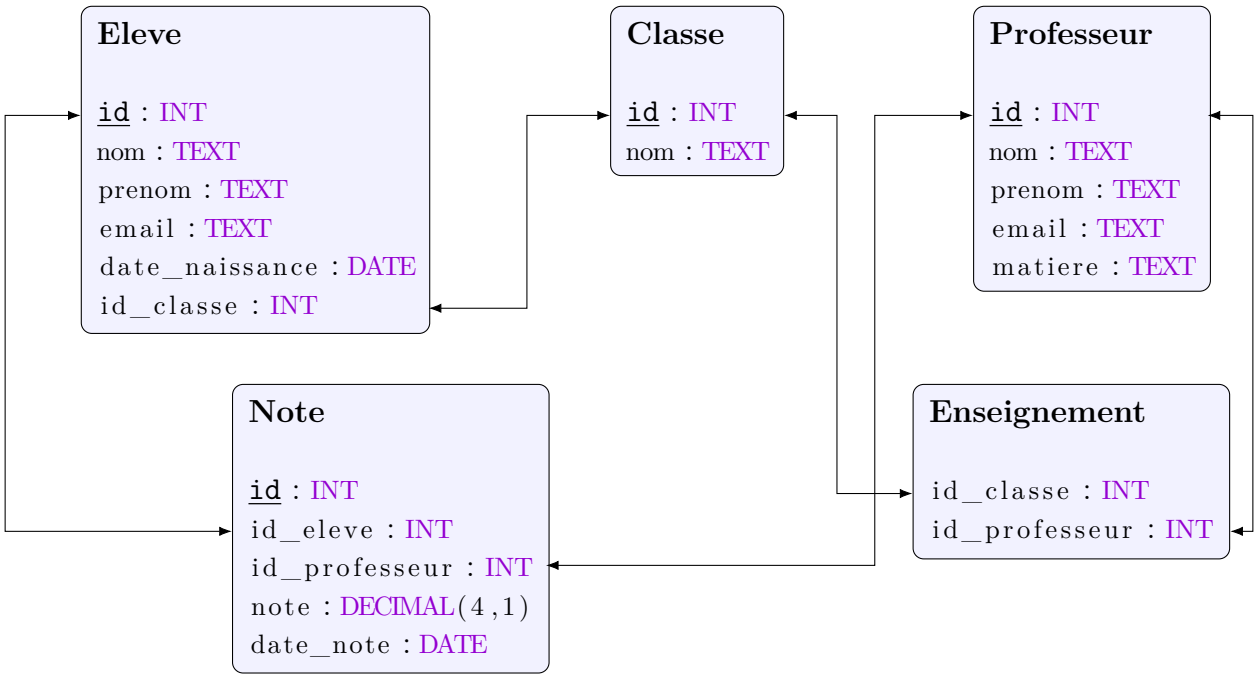

FIGURE 3 – Base de données `lycee` avec les associations

III. Interroger une base de données : requêtes SQL

Nous allons maintenant nous intéresser à la syntaxe des requêtes qui vont nous permettre d'interroger la base de données. Toutes les requêtes que nous allons voir possèdent la même structure :

```
SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ... ORDER BY ... LIMIT ... OFFSET ...;
```

Seuls les deux premiers mots-clés sont obligatoires. Les autres sont optionnels mais, s'ils sont présents, doivent être placés dans cet ordre. Le résultat d'une requête est une nouvelle table (ou relation) qui va être construite à partir des données de la base (prises dans une ou plusieurs des tables initiales).

1) Sélection des attributs et des enregistrements

Projection et sélection

Le premier type de requête est celui de projection/sélection dans une table unique :

```
SELECT [DISTINCT] attributs        -- Projection
FROM table
[WHERE condition];                 -- Sélection
```

- ⋆ FROM table : choix de la table dans laquelle se fait la requête.
- ⋆ SELECT attributs : choix des attributs/colonnes à afficher. Pour afficher toutes les colonnes/attributs, on peut simplement écrire SELECT ∗.
- ⋆ attribut AS alias : chaque attribut peut être renommé.
- ⋆ DISTINCT : supprime les doublons (en cas de lignes identiques sur toutes les colonnes).
- ⋆ WHERE condition : sélection des lignes/enregistrements à afficher à l'aide d'une condition.

Pour écrire une condition, on peut utiliser les opérateurs :

- ⋆ +, −, ∗, / sur les entiers et les flottants
- ⋆ =, <>, <, <=, >, >= sur les entiers, les flottants et les chaînes de caractères (ordre lexicographique)
- ⋆ AND, OR, NOT sur les booléens

Exemple :

```
mysql> SELECT * FROM Professeur;
+----+---------+-----------+-------------------------+-----------------+
| id | nom     | prenom    | email                   | matiere         |
+----+---------+-----------+-------------------------+-----------------+
|  1 | Riou    | Frédéric  | frederic.riou@lycee.fr  | Maths           |
|  2 | Laurent | Nicole    | nicole.laurent@lycee.fr | Physique-Chimie |
|  3 | Paris   | Patrick   | patrick.paris@lycee.fr  | SVT             |
.....................................................................
| 19 | Robert  | Patricia  | patricia.robert@lycee.fr| Français        |
| 20 | Leroux  | Marthe    | marthe.leroux@lycee.fr  | Philosophie     |
+----+---------+-----------+-------------------------+-----------------+
20 rows in set (0.00 sec)
```

```
mysql> SELECT prenom, nom, date_naissance FROM Eleve;
+-----------+---------+----------------+
| prenom    | nom     | date_naissance |
+-----------+---------+----------------+
| Madeleine | Normand | 2007-11-20     |
| Odette    | Regnier | 2010-08-25     |
| Marguerite| Lopes   | 2008-09-19     |
.....................................
| Tristan   | Joseph  | 2009-06-22     |
| Olivier   | Le Gall | 2009-08-11     |
+-----------+---------+----------------+
200 rows in set (0.00 sec)
```

```
mysql> SELECT prenom AS 'Prénom', nom AS 'Nom de famille',
              date_naissance AS 'Date de naissance'
       FROM Eleve;
+-------------+-----------------+--------------------+
| Prénom      | Nom de famille  | Date de naissance  |
+-------------+-----------------+--------------------+
| Madeleine   | Normand         | 2007-11-20         |
| Odette      | Regnier         | 2010-08-25         |
| Marguerite  | Lopes           | 2008-09-19         |
.....................................................
| Tristan     | Joseph          | 2009-06-22         |
| Olivier     | Le Gall         | 2009-08-11         |
+-------------+-----------------+--------------------+
200 rows in set (0.00 sec)
```

```
mysql> SELECT matiere FROM Professeur;
+-----------------+
| matiere         |
+-----------------+
| Maths           |
| Physique-Chimie |
| SVT             |
| Français        |
| Histoire        |
| Géographie      |
| Anglais         |
| Espagnol        |
| Philosophie     |
| Informatique    |
| Philosophie     |
| Physique-Chimie |
| Informatique    |
| Anglais         |
| Maths           |
| Maths           |
| Physique-Chimie |
| Français        |
| Français        |
| Philosophie     |
+-----------------+
20 rows in set (0.00 sec)
```

```
mysql> SELECT DISTINCT matiere
       FROM Professeur;
+-----------------+
| matiere         |
+-----------------+
| Maths           |
| Physique-Chimie |
| SVT             |
| Français        |
| Histoire        |
| Géographie      |
| Anglais         |
| Espagnol        |
| Philosophie     |
| Informatique    |
+-----------------+
10 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Eleve WHERE prenom = 'Paul';
+-----+--------+--------+-----------------------+-----------------+-----------+
| id  | nom    | prenom | email                 | date_naissance  | id_classe |
+-----+--------+--------+-----------------------+-----------------+-----------+
|  17 | Colas  | Paul   | paul.colas@eleves.fr  | 2009-01-03      |         7 |
| 154 | Nicolas| Paul   | paul.nicolas@eleves.fr| 2010-07-16      |         6 |
| 192 | Joly   | Paul   | paul.joly@eleves.fr   | 2009-12-11      |         2 |
| 198 | Pinto  | Paul   | paul.pinto@eleves.fr  | 2009-12-14      |         6 |
+-----+--------+--------+-----------------------+-----------------+-----------+
4 rows in set (0.00 sec)
```

```
mysql> SELECT id, nom, prenom, date_naissance, id_classe
       FROM Eleve
       WHERE date_naissance > '2010-03-10' AND id_classe = 6;
+-----+-----------+----------+-----------------+------------+
| id  | nom       | prenom   | date_naissance  | id_classe  |
+-----+-----------+----------+-----------------+------------+
|  26 | Aubry     | Sabine   | 2010-08-30      |          6 |
|  32 | Chrétien  | Claude   | 2010-08-24      |          6 |
|  49 | Olivier   | Laurent  | 2010-05-14      |          6 |
.....................................................
| 189 | Garcia    | Éric     | 2010-04-13      |          6 |
| 196 | Joly      | Audrey   | 2010-04-15      |          6 |
+-----+-----------+----------+-----------------+------------+
14 rows in set (0.00 sec)
```

Tris des résultats et pagination

- ORDER BY c1 (ASC|DESC), ..., cn (ASC|DESC) : permet de trier les lignes en fonction de la valeur d'un ou plusieurs attributs dans l'ordre croissant (ASC) ou décroissant (DESC). L'ordre croissant est l'ordre par défaut.
- LIMIT n : permet d'afficher les n premières lignes (au plus).
- LIMIT n OFFSET p : permet d'afficher les lignes p + 1 à p + n (au plus). Cela permet d'effectuer une pagination en affichant les résultats d'une requête contenant beaucoup de lignes page par page.

Exemple :

```
mysql> SELECT id, nom, prenom
       FROM Professeur
       ORDER BY nom DESC;
+-----+-----------+-----------+
| id  | nom       | prenom    |
+-----+-----------+-----------+
|  19 | Robert    | Patricia  |
|   1 | Riou      | Frédéric  |
|  11 | Rey       | Corinne   |
|  17 | Pons      | Jacqueline|
|   3 | Paris     | Patrick   |
|   7 | Morvan    | Emmanuel  |
|  10 | Morin     | Martin    |
|   8 | Maury     | Audrey    |
|  12 | Marion    | André     |
|  18 | Lopes     | Suzanne   |
|  20 | Leroux    | Marthe    |
|   2 | Laurent   | Nicole    |
|   6 | Labbé     | Nicole    |
|  16 | Humbert   | Matthieu  |
|   4 | Hubert    | Guillaume |
|  13 | Guilbert  | Louis     |
|   5 | Gallet    | François  |
|   9 | David     | Antoine   |
|  15 | Boutin    | Émilie    |
|  14 | Auger     | Manon     |
+-----+-----------+-----------+
20 rows in set (0.00 sec)
```

```
mysql> SELECT id, nom, prenom
       FROM Professeur
       ORDER BY prenom, nom DESC;
+-----+-----------+-----------+
| id  | nom       | prenom    |
+-----+-----------+-----------+
|  12 | Marion    | André     |
|   9 | David     | Antoine   |
|   8 | Maury     | Audrey    |
|  11 | Rey       | Corinne   |
|  15 | Boutin    | Émilie    |
|   7 | Morvan    | Emmanuel  |
|   5 | Gallet    | François  |
|   1 | Riou      | Frédéric  |
|   4 | Hubert    | Guillaume |
|  17 | Pons      | Jacqueline|
|  13 | Guilbert  | Louis     |
|  14 | Auger     | Manon     |
|  20 | Leroux    | Marthe    |
|  10 | Morin     | Martin    |
|  16 | Humbert   | Matthieu  |
|   2 | Laurent   | Nicole    |
|   6 | Labbé     | Nicole    |
|  19 | Robert    | Patricia  |
|   3 | Paris     | Patrick   |
|  18 | Lopes     | Suzanne   |
+-----+-----------+-----------+
20 rows in set (0.00 sec)
```

```
mysql> SELECT id FROM Professeur
       LIMIT 5;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
|  4 |
|  5 |
+----+
5 rows in set (0.00 sec)
```

```
mysql> SELECT id FROM Professeur
       LIMIT 5 OFFSET 5;
+----+
| id |
+----+
|  6 |
|  7 |
|  8 |
|  9 |
| 10 |
+----+
5 rows in set (0.00 sec)
```

```
mysql> SELECT note FROM Note
       ORDER BY note DESC LIMIT 3;
+------+
| note |
+------+
| 20.0 |
| 20.0 |
| 20.0 |
+------+
3 rows in set (0.01 sec)
```

```
mysql> SELECT DISTINCT note FROM Note
       ORDER BY note DESC LIMIT 3;
+------+
| note |
+------+
| 20.0 |
| 19.9 |
| 19.8 |
+------+
3 rows in set (0.01 sec)
```

2) Fonctions d'agrégation

Fonctions d'agrégation

Une fonction d'agrégation est une fonction qui s'applique sur une colonne d'une table pour effectuer un calcul. Voici la liste des fonctions d'agrégation à connaître :

- ⋆ COUNT(∗) : nombre de lignes
- ⋆ COUNT(DISTINCT attribut) : nombre de valeurs distinctes d'un attribut
- ⋆ MAX(attribut) : valeur maximale d'un attribut
- ⋆ MIN(attribut) : valeur minimale d'un attribut
- ⋆ SUM(attribut) : somme des valeurs d'un attribut
- ⋆ AVG(attribut) : valeur moyenne d'un attribut

Exemple :

```
mysql> SELECT COUNT(*) FROM Eleve
       WHERE id_classe = 1;
+----------+
| COUNT(*) |
+----------+
|       30 |
+----------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(DISTINCT prenom)
       FROM Eleve;
+------------------------+
| COUNT(DISTINCT prenom) |
+------------------------+
|                    132 |
+------------------------+
1 row in set (0.19 sec)
```

```
mysql> SELECT MAX(note) AS Maximum, MIN(note) AS Minimum,
              SUM(note) AS Somme, AVG(note) AS Moyenne
       FROM Note;
+---------+---------+----------+----------+
| Maximum | Minimum | Somme    | Moyenne  |
+---------+---------+----------+----------+
|    20.0 |     0.0 | 267098.2 | 9.93854  |
+---------+---------+----------+----------+
1 row in set (0.01 sec)
```

Remarque : Dans les exemples précédents, on remarque qu'à chaque fois le résultat ne contient qu'une seule et unique ligne. En effet, le calcul se fait à chaque fois sur l'ensemble des enregistrements de la requête. Pour modifier ce comportement, on peut utiliser le mot-clé GROUP BY.

Regroupements des agrégats

Lorsqu'on utilise une ou plusieurs fonctions d'agrégations, on peut ajouter :

- ⋆ GROUP BY `attribut` : permet d'obtenir un résultat de la fonction d'agrégation par groupe de valeurs distinctes de l'attribut.
- ⋆ HAVING `condition` : permet de filtrer les résultats **après calcul de la fonction d'agrégation**.

Exemple :

```
mysql> SELECT id_eleve, AVG(note) AS Moyenne FROM Note
       WHERE date_note BETWEEN '2025-09-01' AND '2025-09-30'
       GROUP BY id_eleve;
+----------+----------+
| id_eleve | Moyenne  |
+----------+----------+
|        1 | 14.57500 |
|        2 |  5.75556 |
|        3 | 11.21667 |
.....................
|      199 | 14.86667 |
|      200 | 14.87857 |
+----------+----------+
200 rows in set (0.04 sec)
```

```
mysql> SELECT id_eleve, AVG(note) AS Moyenne FROM Note
       WHERE date_note BETWEEN '2025-09-01' AND '2025-09-30'
       GROUP BY id_eleve HAVING Moyenne > 10;
+----------+----------+
| id_eleve | Moyenne  |
+----------+----------+
|        1 | 14.57500 |
|        3 | 11.21667 |
|        4 | 11.05385 |
.....................
|      199 | 14.86667 |
|      200 | 14.87857 |
+----------+----------+
99 rows in set (0.04 sec)
```

3) Jointures

Jointure interne

Les jointures permettent de combiner des informations provenant de plusieurs tables. On utilise en général les clés primaires et les clés étrangères pour faire le lien entre les tables. La structure d'une requête contenant une ou plusieurs jointures est la suivante :

```
SELECT attributs
FROM table1
JOIN table2 ON condition
[JOIN table3 ON condition ...];
```

- ⋆ Les conditions après ON sont des conjonctions d'égalités. En général ces conditions sont de la forme

  `clé étrangère de table1 = clé primaire de table2`

- ⋆ Dans `SELECT attributs`, on peut faire référence aux attributs de chacune des tables de la jointure. Si un nom d'attribut est présent dans plusieurs tables, il faut préfixer le nom de l'attribut par le nom de la table : `table.attribut`
- ⋆ Il est possible de renommer les tables à l'aide du mot-clé AS. On peut alors préfixer les attributs à l'aide de l'alias.

Exemple :

```
-- Élèves avec nom de la classe
SELECT e.nom, e.prenom, c.nom AS classe
FROM Eleve AS e JOIN Classe AS c ON e.id_classe = c.id
ORDER BY c.nom, e.nom, e.prenom;

-- Couples (professeur, classe) existants via la table d'association
   Enseignement
SELECT p.nom AS nom_prof, p.prenom AS prenom_prof, p.matiere, c.nom AS classe
FROM Enseignement AS e
JOIN Professeur AS p ON p.id = e.id_professeur
JOIN Classe     AS c ON c.id = e.id_classe
ORDER BY p.nom, c.nom;

-- Notes avec l'élève, le professeur correspondant et la matière
SELECT n.date_note, n.note,
       e.prenom AS prenom_eleve, e.nom AS nom_eleve,
       p.prenom  AS prenom_prof,  p.nom AS nom_prof, p.matiere
FROM Note AS n
JOIN Eleve      AS e ON e.id = n.id_eleve
JOIN Professeur AS p ON p.id  = n.id_professeur
ORDER BY n.date_note DESC
LIMIT 10;
```

Auto-jointure

L'auto-jointure consiste à joindre une table avec elle-même. Elle nécessite de donner des alias distincts :

```
SELECT attributs
FROM table AS alias1
JOIN table AS alias2 ON condition;
```

Exemple :

```
-- Paires d'élèves homonymes (même nom ET même prénom), sans doublon
SELECT e1.id AS id1, e2.id AS id2, e1.nom, e1.prenom
FROM Eleve AS e1
JOIN Eleve AS e2
ON e1.nom = e2.nom AND e1.prenom = e2.prenom
WHERE e1.id < e2.id;

-- Paires de professeurs enseignant la même matière
SELECT p1.nom AS nom1, p1.prenom AS prenom1,
       p2.nom AS nom2, p2.prenom AS prenom2,
       p1.matiere
FROM Professeur AS p1
JOIN Professeur AS p2 ON p1.matiere = p2.matiere
WHERE p1.id < p2.id;
```