Bases De Données

Objectifs

— choix de structures de données,
— notions d'algèbre relationnelle,
— requêtes SQL sur une table,
— requêtes SQL sur plusieurs tables,
— manipulation de BDD avec `Python`.
Pour une simple révision de la syntaxe `SQL`, passer directement aux chapitres 3, 4, 6, 7 et 8.

I Introduction

Choisir une structure de données adaptée au traitement est un problème initial à la réalisation d'un algorithme efficace. Un langage de programmation tel que `Python` propose différents types de données, comme les listes, les tuples ou les dictionnaires (hors du programme d'IPT). Cependant ces types ne sont pas adaptés pour les structures de données non hiérarchiques.

Les structures plates telles que les tableaux ne conviennent pas pour représenter et surtout rechercher certains types d'informations.

1.1 Exemple : gestion des résultats de mes élèves en IPT2

J'ai trois classes de deuxième année cette année en IPT. Je souhaite pouvoir suivre leurs résultats au cours de l'année, mais aussi envoyer ces résultats à leurs professeurs de l'année précédente.

Je veux pouvoir mémoriser pour mes élèves, leurs nom, prénom, et leur provenance, c'est à dire la classe où ils étaient l'année précédente.

Si je prends trois tableaux pour mes trois classes, je facilite l'édition de mes bulletins.

Si je prends un tableau par provenance, je facilite mes envois à mes collègues.

La représentation des données est implicitement choisie pour faciliter le type de recherche prévu. Or la représentation du problème sous forme de tableaux contraint à privilégier un type de lien d'appartenance, ici d'un élève avec sa classe actuelle ou avec sa classe d'origine.

1.2 Un début de réponse : notions d'algèbre relationnelle

L'idée de l'algèbre relationnelle, qui permet de formaliser *les bases de données*, consiste à représenter les informations sous forme de plusieurs tableaux appelés *tables* ou *relations* dans lesquels il n'y a pas de lignes ou de colonnes privilégiées, mais aussi *sans se soucier* des liens d'appartenance.

Dans mon exemple :

On aimerait de plus :
— que la saisie des données soit peu contraignante ;
— pouvoir collecter des informations dans la base de données sans avoir à programmer ;
— pouvoir recouper des informations provenant de plusieurs bases de données ;
— ne pas avoir à se soucier de la façon dont sont stockées les informations ;
— autoriser les accès de différents clients.

II Algèbre relationnelle

2.1 Définitions

LYCEES, DISCIPLINES, ENSEIGNANTS, CLASSES, ELEVES, DEVOIRS et NOTES forment des *relations*, au sens mathématique d'une relation binaire par exemple, ou des *tables* dans le vocabulaires des bases de données.

Note Maths 1. *$\mathcal{R}$, une **relation binaire** entre deux ensembles E et F, est caractérisée par un sous ensemble du produit cartésien $E \times F$. C'est une collection de couples $(x_E,\ x_F)$ où x_E et x_F sont en relation par $\mathcal{R}$.*

On peut alors dire que « MPSI » et « 2 » sont en *relation* par CLASSES.

Note Maths 2. *On considère un ensemble fini $\mathcal{A} = \{A_0,\ \ldots,\ A_{n-1}\}$ dont les éléments sont distincts et appelés **les attributs**. $\mathcal{A} = (A_i)_{0\leq i\leq n-1}$ est donc l'**ensemble des attributs** d'une relation $\mathcal{R}$.*

Par exemple, $\mathcal{A} = \{id\_classe,\ lycee\_classe,\ niveau\_classe,\ filiere\_classe,\ identifiant\_classe\}$ pour la relation CLASSES, où *id_classe* est un de ces *attributs*.

Note Maths 3. *On peut alors définir les **domaines des attributs** $D_i = \mathrm{dom}(A_i)$.*

Pour la relation CLASSES, on a :
— $\mathrm{dom}(id\_classe) = \mathbb{N}^*$ (c'est la clé primaire, on y reviendra plus tard) ;
— $\mathrm{dom}(lycee\_classe) =$ texte ;
— $\mathrm{dom}(niveau\_classe) =$ texte ;
— $\mathrm{dom}(filiere\_classe) =$ texte ;
— $\mathrm{dom}(identifiant\_classe) =$ texte.
On peut alors définir également la *famille des domaines* $\mathcal{D} = \bigcup_{A\in\mathcal{A}} \mathrm{dom}(A)$

Note Maths 4. *Un **schéma relationnel** est un n-uplet de la forme $S = (A_i,\ D_i)_{0\leq i\leq n-1} \in \mathcal{A}^n$ où les A_i sont* distincts deux à deux*.*

Remarque 1. *On oublie parfois les domaines en notant un schéma relationnel $S = (A_i)_{0\leq i\leq n-1}$.*

Par exemple, la table DISCIPLINES est la relation associée au schéma relationnel :

$$S = (id\_discipline,\ nom\_discipline)$$

Elle est composée d'un nombre fini de n-uplets de $\mathrm{dom}(id\_discipline) \times \mathrm{dom}(nom\_discipline)$.

Note Maths 5. *Enfin, les éléments d'une relation sont appelées **valeurs** ou **enregistrements** de cette table. Leur nombre est appelé **cardinal** de la relation.*

On peut alors représenter chaque table par un tableau. La base de donnée sera constituée de toutes ces tables construites sans soucis de liens d'appartenance pour l'instant. Lier ces tables entre elles est le rôle des *opérateurs*.
Voici la table CLASSES complétées pour le lycée Clemenceau (avec l'aide du logiciel `Base` de la suite logicielle `LibreOffice`).

	id_classe	lycee_classe	niveau_classe	filiere_classe	identifiant_classe
▶	0	0	CPGE1	2	1
	1	0	CPGE1	2	2
	2	0	CPGE1	2	3
	3	0	CPGE1	4	1
	4	0	CPGE1	4	2
	5	0	CPGE1	4	3
	6	0	CPGE2	3	
	7	0	CPGE2	3	e
	8	0	CPGE2	5	
	9	0	CPGE2	5	e
	10	0	CPGE2	6	1
	11	0	CPGE2	6	2
	12	0	CPGE2	6	e
	13	0	CPGE1	EXT	
	14	0	CPGE2	EXT	
	15	0	TERM	EXT	

Note Maths 6. *Soit $n \in \mathbb{N}^*$ et $(D_i)_{0\leq i\leq n-1}$ la famille des domaines des attributs $(A_i)_{0\leq i\leq n-1}$ définissant le schéma relationnel $S = (A_i,\ D_i)_{0\leq i\leq n-1}$.*

*Une **relation** ou une **table** de schéma relationnel S, notée $\mathcal{R}$ ou $\mathcal{R}(S)$, sur le produit cartésien $\prod_{i=0}^{n-1} D_i$ est une partie de $\prod_{i=0}^{n-1} D_i$.*

*Si $e \in \mathcal{R}(S)$ et $A \in S$, on note $e \cdot A$ la **composante** du n-uplet e associée à l'attribut A.*

Plus généralement, si $X = (A_p,\ \ldots,\ A_q) \subset S$, on note $e.X = (e \cdot A_p,\ \ldots,\ e \cdot A_q)$.

Dans la mesure où $\mathcal{R}$ est un ensemble, deux valeurs distinctes diffèrent forcément au moins sur un attribut.

2.2 Base de données

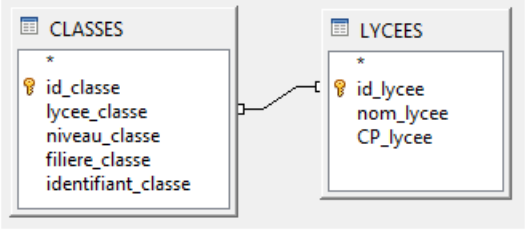
Une *base de données* est un ensemble structuré de données informatiques dans lequel :
- les données sont enregistrées sur un support permanent ;
- les données ne figurent qu'une seule fois (pas de redondance d'information) ;
- chaque objet possède un identifiant unique.

Note Maths 7. *Une **base de données relationnelle** est une base de données constituée d'un ensemble de relations ou tables. Le schéma d'une base de données relationnelle est l'ensemble des schémas des relations qui forment la base de données.*

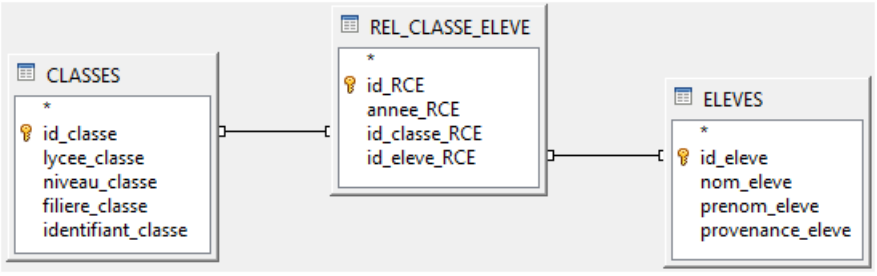
Clé primaire Elle permet d'identifier un enregistrement en ne précisant qu'un unique attribut ou un n-uplet unique d'attributs. Il est préférable de l'ajouter pour chaque table avec une incrémentation automatique.

Grâce à cette clé, il est possible de créer des liens entre plusieurs relations. Dans la table `CLASSES` ci-dessus, l'attribut `lycee_classe` fait apparaître le nombre 0 car c'est la clé primaire qui identifie le lycée Clemenceau.

Le lien entre ces deux tables est représenté dans le logiciel `Base` par un lien entre l'attribut et la clé primaire.



Relation entre deux tables Lorsqu'il est nécessaire de relier deux tables entre elles, on peut créer des relations supplémentaires. Pour notre exemple, il faut relier la table des `ELEVES` avec la table des `CLASSES`.



Dans le cas de cette base de données, il reste encore plusieurs tables de liaison de ce type à construire.

2.3 Opérateurs sur le modèle relationnel et leur syntaxe SQL

Avec les tables précédentes, on souhaite rechercher les noms des élèves qui sont d'anciens élèves de MPSI2 et de PCSI1. Pour pouvoir effectuer cette recherche, on doit pouvoir être capable de :

— sélectionner dans la relation ELEVES les valeurs dont l'attribut `provenance_eleve` correspond aux identifiants de la MPSI2 et de la PCSI1 du lycée ;
— regrouper ces deux ensembles de valeurs ;
— extraire de chacune des valeurs, l'attribut `nom`.

On peut donc décomposer cette recherche à l'aide de trois opérations génériques agissant sur ces relations :

— une *sélection* permettant d'extraire une sous-relation dont les valeurs vérifient un critère donné ;
— une *union* pour regrouper les valeurs de deux relations, sous réserve que les deux relations aient le même schéma ;
— une *projection* pour ne conserver d'une relation que la valeur de certains attributs.

On appelle de telles opérations des *opérateurs relationnels*.

2.3.1 Gestionnaires de bases de données

Il existe plusieurs logiciels de gestion de bases de données qui utilisent tous le même langage SQL pour *Structured Query Language* pour effectuer les recherches sur une base de données. Ces requêtes de recherche SQL seront données au fur et à mesure de la découverte des opérateurs.

2.3.2 Projection π

Note Maths 8. *Soit $\mathcal{R}(S)$ une relation de schéma S et $X \subset S$ un sous n-uplet de S. On appelle* ***projection*** *de $\mathcal{R}$ selon X la relation :*

$$\pi_X(\mathcal{R}) = \{e(X) \mid e \in \mathcal{R}\}$$

Le schéma relationnel de $\pi_X(\mathcal{R})$ est donc X. C'est l'ensemble des attributs conservés par la projection. Une projection ne contiendra donc pas forcément autant de valeurs que la relation de départ car plusieurs valeurs peuvent être fusionnées.

Par exemple, $\pi_{filiere\_classe}(CLASSES)$ va rendre un tableau avec seulement six lignes : MPSI, PCSI, MP, PC et PSI.

SQL Une projection $\pi_{A_p,\ ...,\ A_q}(\mathcal{R})$ sera effectuée par la commande : `SELECT A_p, ..., A_q FROM R ;`
Soit pour notre exemple : `SELECT filiere_classe FROM Classes ;`
Quelques remarques sur l'écriture de requête SQL :

— Toutes les recherches de requête SQL démarrent pas `SELECT` et se terminent par un point-virgule.
— `SELECT` est un faux-ami, il n'effectue pas une sélection, mais une projection.
— La relation sur laquelle opère la requête est précisée par `FROM`.
— Les requêtes SQL ne sont pas sensibles à la casse, mais il est d'usage de mettre les mots-clés en majuscules et les attributs en minuscules. Personnellement, je prends également l'usage de mettre les noms de tables avec une majuscule. Les valeurs des attributs non numériques sont écrites entre guillemets simples ou doubles.
— Il est possible d'obtenir l'intégralité de la relation $\mathcal{R}$ avec la requête `SELECT * FROM R ;`.

2.3.3 Sélection σ

Note Maths 9. *Si $\mathcal{R}(S)$ est une relation de schéma relationnel S, $A \in S$ un attribut de $\mathcal{R}(S)$, et $a \in \mathrm{dom}(A)$ une valeur de cet attribut, on appelle* ***sélection de $\mathcal{R}$ selon $A = a$*** *la relation obtenue en sélectionnant dans $\mathcal{R}$ uniquement les valeurs e telles que $e \cdot A = a$. On la note $\sigma_{A=a}(\mathcal{R})$. On a donc :*

$$\sigma_{A=a}(\mathcal{R}) = \{e \in \mathcal{R} \mid e \cdot A = a\}$$

Si le domaine de l'attribut le permet, on peut étendre la notion de sélection à d'autres comparaisons que l'égalité.

Par exemple $\sigma_{10 \leqslant valeur\_note \leqslant 12}(Notes)$ sélectionnera toutes les notes comprises entre 10 et 12.

SQL Une sélection $\sigma_{A=a}(\mathcal{R})$ sera effectuée par la commande `SELECT * FROM R WHERE A = a ;`
Pour notre exemple, cela donne :

```
SELECT * FROM Notes WHERE valeur_note >= 10 AND valeur_note <= 12 ;
```

Il est donc possible d'utiliser les opérateurs logiques `AND`, `OR` et `NOT` pour construire ses requêtes.

2.3.4 Opérateurs ensemblistes usuels et sélections composées

Si deux relations ont le même schéma, il est possible de leur appliquer des opérateurs ensemblistes comme l'union $\cup$, l'intersection $\cap$ et la différence $\setminus$. La différence entre deux relations $\mathcal{R}_1(S)$ et $\mathcal{R}_2(S)$ est l'ensemble des valeurs comprises dans $\mathcal{R}_1$, mais non dans $\mathcal{R}_2$. On la note $\mathcal{R}_1 \setminus \mathcal{R}_2$ de schéma S.

Avec les opérateurs ensemblistes, il est possible d'exprimer des sélections complexes.

Remarque 2. *La contrainte de ces opérateurs sur la compatibilité des schémas relationnels est de fait levée si on travaille sur deux sélections d'une même relation.*

Note Maths 10. *Si on note $\mathcal{R}_1$ et $\mathcal{R}_2$ deux sélections de $\mathcal{R}$ satisfaisant respectivement les conditions C_1 et C_2 :*

- *pour obtenir les valeurs vérifiant $C = C_1$ ET C_2, on calcule $\mathcal{R}_1 \cap \mathcal{R}_2$;*
- *pour obtenir les valeurs vérifiant $C = C_1$ OU C_2, on calcule $\mathcal{R}_1 \cup \mathcal{R}_2$;*
- *pour obtenir les valeurs vérifiant $C = C_1$ ET NON C_2, on calcule $\mathcal{R}_1 \setminus \mathcal{R}_2$.*

Par exemple, $\sigma_{filiere\_classe = 'PSI'}(Classes) \cap \sigma_{identifiant\_classe\ != 'e'}(Classes)$ va donner les deux classes non étoilées de PSI du lycée.

SQL Pour réaliser des opérations ensemblistes, il est d'abord nécessaire d'obtenir l'intégralité des tables grâce à une projection avec l'attribut `*`. Ensuite, on utilise les mots-clés `UNION`, `INTERSECT` ou `EXCEPT` pour réaliser les opérations d'union, d'intersection ou de différence.

Cela donne :

```
SELECT * FROM Classes WHERE filiere_classe = 'PSI' INTERSECT
 SELECT * FROM Classes WHERE identifiant_classe != 'e' ;
```

2.3.5 Renommage ρ

Il est possible de renommer un attribut d'une relation à l'aide d'un opérateur dit de *renommage*. Ce changement de nom ne modifiera pas les valeurs de la relation car les attributs n'apparaissent que par l'intermédiaire de leur domaine.

Note Maths 11. *Soit $S = (A_0,\ \ldots,\ A_{n-1})$ un schéma relationnel, $i \in [\![0\ ;n-1]\!]$ et B un attribut tel que $\mathrm{dom}(B) = \mathrm{dom}(A_i)$. On note le schéma déduit de S en **renommant** A_i en B :*

$$\rho_{A_i \leftarrow B}(S) = (A_0,\ \ldots,\ A_{i-1},\ B,\ A_{i+1},\ \ldots,\ A_{n-1})$$

SQL Un renommage $\rho_{A_i \leftarrow B}(S)$ sera effectué avec le mot-clé `AS` par la commande :

```
SELECT A_0, ..., A_{i-1}, A_i AS B, A_{i + 1}, ..., A_{n-1} FROM R ;
```

Il est donc nécessaire d'indiquer tous les attributs.

2.3.6 Agrégation

L'agrégation va effectuer deux choses. Elle va d'abord servir à regrouper les valeurs d'une relation selon les éléments du domaine d'un attribut choisi, $\mathrm{dom}(A)$ par exemple. Ces groupes de valeurs seront des *agrégats*. Ensuite, elle va appliquer une fonction aux valeurs d'un autre attribut, B par exemple, notée $f(B)$ sur chacun de ces agrégats. On la note :

$$_A\gamma_{f(B)}(R)$$

SQL Les cinq fonctions d'agrégation et leur syntaxe SQL sont données ci-dessous.

Algèbre relationnelle	SQL
comptage	`COUNT`
max	`MAX`
min	`MIN`
somme	`SUM`
moyenne	`AVG`

Pour illustrer cette dernière section, voici la table `DEVOIRS` suivie de la table `NOTES`.

	id_devoir	classe_devoir	discipline_devoir	enseignant_devoir	date_devoir	nom_devoir
▶	0	10	9	2	16.03.15	Sujet synthese
	1	1	15	2	08.12.14	SI_2
	2	1	9	3	05.01.15	IPT_1
	3	1	15	2	13.10.14	SI_1
✱	<AutoCham					

	id_note	devoir_note	eleve_note	valeur_note
	10	1	17	13
	11	1	18	15
	12	1	19	11
	13	1	20	5
	14	1	21	9
	20	2	17	9
	21	2	18	12
	22	2	19	12
	23	2	20	16
	24	2	21	
	30	3	17	11
	31	3	18	16
	32	3	19	13
	33	3	20	11
▶	34	3	21	8
✱	<AutoChar			

Quelques exemples de requêtes SQL :

1. Pour effectuer une agrégation simple du type $\gamma_{f(A)}(R)$: `SELECT f(A) FROM R ;`
   Par exemple, pour connaître la moyenne des notes : `SELECT AVG(valeur_note) FROM Notes ;`
2. Si on souhaite regrouper ces notes par devoirs avant d'en effectuer la moyenne, il faut faire une agrégation du type ${}_{devoir\_note}\gamma_{moyenne(valeur\_note)}(Notes)$, soit en langage SQL :
   `SELECT devoir_note, AVG(valeur_note) FROM Notes GROUP BY devoir_note ;`
3. Enfin, si on souhaite obtenir le nombre d'élèves qui ont eu la même note lors d'un même devoir, il faut faire une agrégation du type ${}_{devoir\_note,valeur\_note}\gamma_{comptage(eleve\_note)}(Notes)$, soit en langage SQL :
   ```
   SELECT devoir_note, valeur_note, COUNT(eleve_note) FROM Notes
    GROUP BY devoir_note, valeur_note ;
   ```

III Récapitulatif des commandes SQL mono-tables

3.1 Structure de la table

— Création d'une table :
`CREATE TABLE Nom_table (col1 type, col2 type, ...) ;`
Les types utilisés sont par exemple `INTEGER`, `REAL` ou `TEXT`.
On créera toujours une colonne `id INTEGER PRIMARY KEY`.
— Ajout d'une colonne :
`ALTER TABLE Nom_table ADD COLUMN nom_colonne type ;`

3.2 Gestion des entrées

— Nouvelle entrée :
`INSERT INTO Nom_table (col1, col2, ...) VALUES (val1, val2, ...) ;`
Les colonnes non précisées reçoivent `NULL`, sauf la colonne de clé primaire qui est incrémentée automatiquement.
— Modifications d'entrées :
`UPDATE Nom_table SET col1 = val1, col2 = val2, ... WHERE conditions ;`
Si on ne précise pas de condition, toutes les entrées sont modifiées.
— Suppression d'entrées :
`DELETE FROM Nom_table WHERE conditions ;`
Si on ne précise pas de condition, toutes les entrées sont supprimées.

3.3 Affichage, sélection, projection

— Requête élémentaire :

```
SELECT col1, col2, ... FROM Nom_table WHERE conditions ;
```
Si on ne précise pas de condition, toutes les entrées sont renvoyées. Si on remplace la liste des colonnes par *, toutes les colonnes de la table sont renvoyées.

— Élimination des doublons :
Si on ne souhaite pas répéter les lignes qui ont le même enregistrement pour un attribut, il faut ajouter le mot-clé `DISTINCT`.
```
SELECT col1, DISTINCT col2, ... FROM Nom_table WHERE conditions ;
```

— Données extraites et données dérivées :
Pour afficher autre chose que le nom d'une colonne, soit par un calcul de valeurs issues de différentes colonnes, soit en donnant un autre nom à la colonne résultat pour affichage. L'exemple ci-dessous affiche le prix d'un produit diminuée de sa ristourne.
```
SELECT nom_produit, prix, ristourne, prix * (1 - ristourne / 100) AS prix_reduit
 FROM Produits WHERE conditions ;
```
Il existe plusieurs fonctions `SQL` qui permettent de dériver des valeurs à partir des valeurs des colonnes, sur les chaînes de caractères aussi, des fonctions de conversion de type, temporelles, de sélection...

3.4 Conditions

— Opérateurs de comparaison : `=`, `!=`, `<`, `<=`, `>`, `>=`.
— Opérateurs logiques : `AND`, `OR`, `NOT`.
— Conditions particulières :
  — `col IS NULL` / `col IS NOT NULL`, pour tester la présence d'un enregistrement pour l'attribut `col`.
  — `val IN ('val1', 'val2', ... )`, pour tester l'appartenance à une liste.
  — `val BETWEEN val1 AND val2`, `cat NOT BETWEEN 'B2' AND 'D4'`, les caractères sont ordonnés par leurs codes ASCII (alors `'A' < 'a'` car `ord('A') = 65` et `ord('a') = 97`).
  — `cat LIKE '_1'`, recherche parmi les mots de deux carcatères dont le deuxième est '1'.
  — `adresse LIKE '%rue%'`, recherche parmi les chaînes de carcatères contenant le mot 'rue'.
  — `date LIKE '%2016%'`, recherche parmi les dates contenant l'année '2016'.
— Conditions d'association ou Sous-requête :
```
SELECT col1, col2, ... FROM Nom_table
WHERE col3 IN (SELECT col3 FROM Nom_table WHERE conditions) ;
```
Le résultat d'une requête peut servir de liste de valeurs pour une condition.
```
SELECT col1, col2, ... FROM Nom_table
WHERE col3 = (SELECT col3 FROM Nom_table WHERE conditions) ;
```
Si le résultat du `SELECT` ne contient qu'une valeur (une colonne, une entrée), on peut utiliser `=` au lieu de `IN`.

3.5 Opérateurs ensemblistes

— Ajouter deux résultats de requêtes :
```
(SELECT ... FROM ... WHERE ...) UNION (SELECT ... FROM ... WHERE ...) ;
```
— Récupérer les valeurs communes à deux requêtes :
```
(SELECT ... FROM ... WHERE ...) INTERSECT (SELECT ... FROM ... WHERE ...) ;
```
— Afficher les valeurs qui appartiennent à une première requête mais qui n'appartiennent pas à une seconde :
```
(SELECT ... FROM ... WHERE ...) EXCEPT (SELECT ... FROM ... WHERE ...) ;
```
Il est à noter qu'il est possible d'obtenir le même résultat avec des jointures (voir la section sur les requêtes multi-tables).

Les valeurs `NULL` sont ignorées dans les calculs.

3.6 Fonctions d'agrégation

— Dénombrer les entrées de la table :
```
SELECT COUNT(*) FROM Nom_table WHERE conditions ;
```
— Dénombrer sans répétitions :
```
SELECT COUNT(DISTINCT colonne) FROM Nom_table WHERE conditions ;
```
— Minimum d'un attribut :
```
SELECT MIN(colonne) FROM Nom_table WHERE conditions ;
```
— Maximum d'un attribut :
```
SELECT MAX(colonne) FROM Nom_table WHERE conditions ;
```
— Somme des attributs :
```
SELECT SUM(colonne) FROM Nom_table WHERE conditions ;
```

— Moyenne des attributs :
`SELECT AVG(colonne) FROM Nom_table WHERE conditions ;`

3.7 Extraction de données groupées

Pour illustrer cette partie et la suivante, considérons une table `Eleves` formée de 4 colonnes : `id_eleve`, `nom_eleve`, `classe` et `note`.

```
SELECT classe, COUNT(*) AS 'Nombre d'élèves', AVG(Note) AS 'Moyenne de classe'
   FROM Eleves GROUP BY classe ;
```

Cette requête permet d'afficher pour chaque classe, le nombre d'élèves et la moyenne de classe. Pour pouvoir être regroupées, les projections doivent fournir une seule valeur par groupe, sinon la clause `GROUP BY` ne peut être appliquée.

À présent, il est aussi possible de poser une condition avant d'effectuer ces regroupements par la nouvelle clause `HAVING`.

```
SELECT classe, COUNT(*) AS 'Nombre d'élèves', AVG(Note) AS 'Moyenne de classe'
   FROM Eleves GROUP BY classe HAVING COUNT(*) >= 40 ;
```

Cette nouvelle requête affiche le même résultat que la requête précédente, mais seulement pour les classes de plus de 40 élèves.

Les critères de regroupement peuvent se composer.

3.8 Ordonnancement des lignes résultats d'une requête

```
SELECT nom_eleve, note FROM Eleves WHERE classe = 'MPSI2'
   ORDER BY note ;
```

L'affichage se fera dans le sens des notes croissantes.

Pour obtenir un classement avec les notes décroissantes, il faut écrire `ORDER BY note DESC`.

Pour composer les critères d'ordonnancement, il faut écrire `ORDER BY crit1, crit2, ....` Alors `crit1` sera pris en considération avant `crit2`...

Enfin, `crit i` peut être le nom d'une colonne ou son alias.

IV Application 1 : gestion d'une petite bibliothèque

Voici le schéma d'une table d'une petite bibliothèque contenant les œuvres au programme de français philosophie en classes préparatoires scientifique.

id	Titre	Nom_Auteur	Prenom_Auteur	Nationalite_Auteur
	Annee_publication	Genre	Annee_Concours	Theme

Questionnaire :

1. Créer la table `Oeuvres` ci-dessus.
2. Insérer dans votre table les nouvelles œuvres au programme cette année.
3. Ajouter une colonne Commentaires.
4. Mettre à jour les commentaires pour les œuvres que vous avez lues.
5. Supprimer de la bibliothèque l'ouvrage de Vladimir Jankélévitch.
   À présent, on considère que la table complétée est à votre disposition.
6. Afficher toute la table.
7. Afficher tous les titres des livres de Eschyle.
8. Afficher tous les auteurs de la bibliothèque.
9. Afficher les auteurs, les titres et les années de publication des livres parus depuis la publication des "Lettres persanes" de Montesquieu, ordonnés par année de publication croissante. La première ligne affichera les Lettres persanes.
10. Donner le nombre d'œuvres de Molière.
11. Donner le nombre d'œuvres d'auteurs qui n'ont pas de prénom.
12. Donner l'année moyenne de publication des ouvrages des auteurs français.
13. Donner le nombre de livres et le nombre d'auteurs de la bibliothèque.
14. Donner la valeur moyenne des livres de la bibliothèque.

15. Donner le titre et l'auteur du livre le plus ancien de la bibliothèque.
16. Afficher les nationalités des auteurs et le nombre d'ouvrages d'auteurs de ces nationalités. L'affichage se fera dans l'ordre décroissant.

Des solutions pour ces requêtes sont fournies à la dernière partie de ce polycopié.

V Opérateurs complexes entre plusieurs tables

Pour l'instant les opérateurs définis ne s'appliquent que sur une seule relation, ou éventuellement pour les opérateurs ensemblistes, sur deux relations ayant le même schéma relationnel. Cette section présente de nouveaux opérateurs qui pourront s'appliquer sur plusieurs tables.

Note Maths 12 (Définition)**.** *Si $R(S)$ et $R'(S')$ sont deux relations de schémas disjoints, leur* produit cartésien *est :*

$$R \times R' = \{(v_0,\ \ldots,\ v_{n-1},\ v'_0,\ \ldots,\ v'_{m-1} \mid (v_1,\ \ldots,\ v_{n-1}) \in R \text{ et } (v'_1,\ \ldots,\ v'_{m-1}) \in R'\}$$

5.1 Jointure

Un opérateur de jointure sert à recoller deux relations de **schémas disjoints**.

Note Maths 13 (Définition)**.** *Soient $R(S)$ et $R'(S')$ deux relations de schémas disjoints, et $A \in S$, $A' \in S'$ tels que* $\mathrm{dom}(A) = \mathrm{dom}(A')$*. On note la* jointure symétrique *de S et S' selon $(A,\ A')$:*

$$R[A = A']R' = \{e \in R \times R' \mid e \cdot A = e \cdot A'\} = \sigma_{A=A'}(R \times R')$$

SQL La jointure symétrique s'écrit avec les mots-clés `JOIN` et `ON` :

```
SELECT * FROM R JOIN R' ON A = A' ;
```

Par exemple, si on souhaite avoir la table des différentes classes dans les lycées référencés, il faut effectuer la jointure des tables `CLASSES` et `LYCEES` pour `[lycee_classe = id_lycee]`.

```
SELECT * FROM "CLASSES" JOIN "lycee" ON "lycee_classe" = "id_lycee" ;
```

Le résultat avec `Base` est affiché ci-dessous.

	id_classe	lycee_classe	niveau_classe	filiere_classe	identifiant_cl...	id_lycee	nom_lycee	CP_lycee
▶	0	0	CPGE1	MPSI	1	0	Clemenceau	44042
	1	0	CPGE1	MPSI	2	0	Clemenceau	44042
	2	0	CPGE1	MPSI	3	0	Clemenceau	44042
	3	0	CPGE1	PCSI	1	0	Clemenceau	44042
	4	0	CPGE1	PCSI	2	0	Clemenceau	44042
	5	0	CPGE1	PCSI	3	0	Clemenceau	44042
	6	0	CPGE2	MP		0	Clemenceau	44042
	7	0	CPGE2	MP	e	0	Clemenceau	44042
	8	0	CPGE2	PC		0	Clemenceau	44042
	9	0	CPGE2	PC	e	0	Clemenceau	44042
	10	0	CPGE2	PSI	1	0	Clemenceau	44042
	11	0	CPGE2	PSI	2	0	Clemenceau	44042
	12	0	CPGE2	PSI	e	0	Clemenceau	44042
	13	0	CPGE1	EXT		0	Clemenceau	44042
	14	0	CPGE2	EXT		0	Clemenceau	44042
	15	0	TERM	EXT		0	Clemenceau	44042

Les deux attributs `lycee_classe` et `id_lycee` vont perdurer après jointure et seront redondants. Si besoin on peut éliminer cette redondance avec une projection. Mais cette étape de jointure est le plus souvent une étape intermédiaire pour une requête plus complexe.

Si on souhaite réaliser plusieurs jointures $[R[A = A']R'[B' = B'']R''$:

```
SELECT * FROM R JOIN R' ON A = A' JOIN R'' ON B' = B'' WHERE conditions ;
```

ou `SELECT * FROM R, R', R'' WHERE A = A' AND B' = B'' AND conditions;`

5.2 Auto-jointure

C'est un cas particulier de jointure lorsque la jointure doit se faire entre deux attributs de la même table.

5.3 Composition de requêtes complexes

Tous ces opérateurs peuvent se combiner pour exprimer des requêtes très élaborées, faisant intervenir plusieurs relations. Leur écriture est alors plus délicate. En particulier, la question de l'ordre dans lequel composer les différentes opérations devient alors une vraie question de recherche.

5.4 Exemples de jointures complexes

À partir de la base de données sur mes classes en première page, voici quelques exemples de requêtes complexes avec la table `ELEVES` à laquelle a été ajouté l'attribut `classe_eleve`, plutôt que d'utiliser la table `REL_CLASSE_ELEVE` en troisième page.

5.4.1 Utilisation des jointures

1. On souhaite avoir un tableau affichant les Noms, Prénoms, Filière et Classe des élèves de PCe.
2. On souhaite avoir le tableau affichant les Noms, Prénoms, Filière, Classe, DS et Note des élèves de MPSI2 qui ont eu strictement plus de 10 au devoir 'SI_1'.
   Dans la mesure du possible, il faut toujours réaliser les sélections en amont avec le mot-clé `WHERE`, car cela limite le nombre de valeurs à considérer dans l'agrégation.
   Cependant, lorsque l'on désire sélectionner en fonction du résultat des fonctions d'agrégation, il est obligatoire de le faire en aval à l'aide d'un mot-clé `HAVING`, ce qui est le cas pour la requête demandée ici.
   Utilisation des **agrégations**
3. Afficher la moyenne des devoirs en deux colonnes DS et moyenne.
4. Sélectionner les devoirs où la moyenne est supérieur à 11.
5. Moyenne des devoirs de la MPSI2 sur les colonnes suivantes : Filière, Classe, Discipline, DS, Moyenne.

Des solutions de requêtes sont fournies en dernière partie.

VI Récapitulatif des commandes SQL multi-tables

6.1 Syntaxes d'une jointure simple

— Syntaxe mentionnée au programme d'IPT
```
SELECT col1-1, col2-1, ... FROM Table1 JOIN Table2
   ON (Table1.col = Table2.col) WHERE condition ;
```
`col1-1, col2-1` ne pose pas d'ambiguïté dans les noms des colonnes, mais si deux colonnes des deux tables ont le même nom, il faut utiliser une syntaxe du type `Table1.col = Table2.col`.
`(Table1.col = Table2.col)` est appelé la *condition de jointure*.
Pour éviter la longueur de l'écriture `Table1.col`, il est utile d'utiliser les alias :
`FROM Table1 AS T1 JOIN Table2 AS T2 ON (T1.col = T2.col)`.

— Autre syntaxe plus concise et assez courante
```
SELECT col1-1, col2-1, ... FROM Table1 AS T1, Table2 AS T2
   WHERE (Table1.col = Table2.col) AND condition ;
```
L'ordre d'écriture des tables après la clause `FROM` est indifférent.

— Une jointure particulière `SELECT col1-1, col2-1, ... FROM Table1 NATURAL JOIN Table2`
```
   WHERE condition ;
```
Ici, pas besoin de spécifier la condition de jointure avec l'extension `NATURAL JOIN`, mais il faut que les deux tables ne partagent qu'une et une seule colonne de même nom (par exemple `id`). C'est le cas pour les tables du sujet CCS d'informatique de 2015.

6.2 Syntaxes d'une jointure sur plus de deux tables

— Syntaxe mentionnée au programme d'IPT
```
SELECT col1-1, col2-1, ... FROM Table1 AS T1 JOIN Table2 AS T2
   ON (T1.col = T2.col) JOIN Table3 AS T3 ON (T3.attribut = T2.attribut)
   WHERE condition ;
```
Ici, l'ordre de déclaration des jointures dans la clause `FROM` est important pour les conditions de jointure.

— Autre syntaxe `SELECT col1-1, col2-1, ... FROM Table1 AS T1, Table2 AS T2, Table3 AS T3`
```
   WHERE (T3.attribut = T2.attribut AND T1.col = T2.col) AND condition ;
```
L'ordre d'écriture des jointures ici est indifférent.

6.3 Auto-jointure

Une auto-jointure permet de réaliser des requêtes sur des structures de données cycliques.

Imaginons par exemple une table comportant les noms du personnel d'une entreprise. La table `Personne` contient les trois colonnes suivantes : `id`, `nom` et `id_responsable`. La dernière colonne renvoie donc une référence vers la même table puisque les responsables font eux aussi partie du personnel de l'entreprise.

```
SELECT S.nom AS 'Subordonné', R.nom AS 'Responsable', R.id AS 'id Responsable'
   FROM Personne AS S, Personne AS R WHERE S.responsable = R.id ;
```

Cette requête affiche toutes les personnes qui ont un responsable et affichera leur nom, celui de leur responsable direct, et l'identifiant de leur responsable.

6.4 Réaliser une jointure externe

Si lors d'une jointure entre deux ou plusieurs tables, certaines lignes d'une table n'ont pas de correspondantes dans l'autre table, ces lignes sont dites *célibataires*.

Ces lignes n'apparaîtront pas dans une jointure classique.

Voici deux solutions pour les afficher si besoin.

—
```
SELECT col1-1, col1-2, col2-2 FROM T1, T2 WHERE col1-1 = col2-1
   UNION SELECT NULL, col1-2, col2-2 FROM T1, T2 WHERE col2-1 NOT IN (
   SELECT col1-1 FROM T1) ;
```

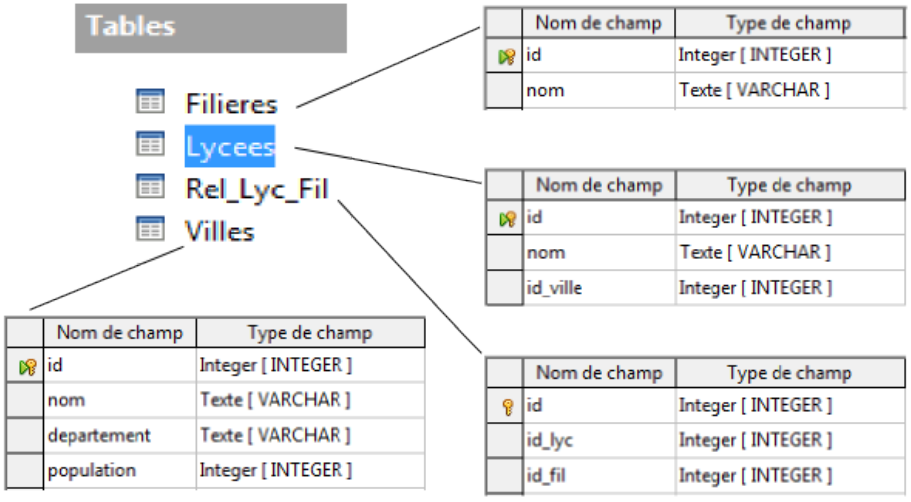
—
```
SELECT col1-1, col1-2, col2-2 FROM T1 RIGHT OUTER JOIN T2
   ON col1-1 = col2-1 ;
```

Les deux requêtes ajoutent les lignes célibataires de T2 à toutes celles qui sont dans la jointure (T1, T2). `OUTER JOIN` réalise une jointure externe avec la table de droite, c'est à dire `T2`.

VII Application 2 : gestion pédagogique des classes préparatoires

La base de données qui a servi à illustrer cette présentation est très complexe et nécessite de créer des tables de relations entre tables, comme vu à la section 2.2

Pour s'entraîner à présent sur les syntaxes `SQL`, en voici une version plus concise et plus simple, présentée sur le schéma suivant.



7.1 Jointure à deux tables

1. Afficher les noms et villes des lycées de Loire-Atlantique (département 44).
2. Afficher les noms des lycées situés dans une ville de moins de 250 000 habitants.
3. Afficher le nombre de lycées situés à Nantes.
4. Afficher le département de la ville du lycée Bergson.

7.2 Jointure à trois tables et plus

1. Afficher les noms des lycées et filières proposées dans chacun d'entre eux.
2. Afficher les noms des lycées proposant une PCSI.
3. Afficher les noms des filières proposées au lycée Clemenceau de Nantes.
4. Ouvrir une PTSI au lycée Clemenceau de Nantes.
5. Fermer cette PTSI.
6. Afficher les filières (avec lycées et villes correspondantes) proposées dans la Sarthe (département 72).
7. Afficher les noms des villes où l'on peut trouver une filière PSI.

8. Afficher les noms des filières proposées dans les villes de moins de 250 000 habitants.

Des solutions de requêtes sont fournies en dernière partie.

VIII Application 3 : Élevage de Pokemons

Pour illustrer les auto-jointures, voici une table utilisée pour gérer un petit élevage de Pokemons.

La table 'Pokemons' possède les attributs suivants : `id` (la clé primaire), `nom`, `sexe`, `niveau`, `points_vie`, `taille_cm`, `poids_kg`, `annee_naissance`, `annee_mort`, `annee_vente`, `idmere` et `idpere`.

1. Afficher les noms des enfants du pokemon femelle "Ludicolo", ainsi que les noms de leurs pères respectifs.
2. Afficher les noms des enfants et petits-enfants de "Ludicolo".
3. Afficher les noms des pokemons frères entiers.
4. Afficher le nombre de pokemons morts avant leurs deux parents (ou en même temps).

Des solutions de requêtes sont fournies en dernière partie.

IX Manipulation de bases de données en Python

```
from sqlite3 import *

mabase = connect('fichierbase.db')  # Ouverture du fichier contenant la base de données

cur = mabase.cursor()  # Création d'un objet curseur permettant d'accéder à la base

# La commande execute prend en entrée une requête SQL sous forme de chaîne de caractères
cur.execute("SELECT nom FROM Pokemon WHERE sexe = 'M';")

# Dans le cas d'un SELECT, le résultat est stocké dans le curseur, on utilise fetchall()
# ou fetchone() pour le récupérer, fetchone() renvoie un tuple correspondant à une entrée
# à chaque appel (à la manière de readline),
# fetchall() renvoie directement une liste de tuples.

cur.execute("SELECT nom, niveau FROM Pokemon WHERE niveau >= 4 ;")
res = cur.fetchall()
nom = []
niveau = []
for entree in res :
    nom.append(entree[0])
    niveau.append(entree[1])



cur.execute("SELECT nom, niveau FROM Pokemon WHERE niveau >= 4 ;")
nom = []
niveau = []
entree = cur.fetchone()
for entree in res :
    nom.append(entree[0])
    niveau.append(entree[1])
    entree = cur.fetchone()

mabase.commit()  # Validation de la transaction en cours, rollback() permet de l'annuler

mabase.close()  # Fermeture de la base de données et enregistrement des résultats.
```

X Corrigés des applications

10.1 Application 1 : gestion d'une petite bibliothèque

1. ```
   CREATE TABLE "Oeuvres" (
       "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
       "Titre" TEXT NOT NULL,
       "Nom_Auteur" TEXT NOT NULL,
       "Prenom_Auteur" TEXT,
       "Nationalite_Auteur" TEXT,
       "Annee_publication" INTEGER,
       "Genre" TEXT,
       "Annee_Concours" INTEGER NOT NULL,
       "Theme" TEXT NOT NULL);
   ```
2. ```
   INSERT INTO "oeuvres" VALUES (NULL,
   "L'aventure, l'ennui, le sérieux", "Jankélévitch", "Vladimir", "Français", 1963,
   "Essai philosophique", 2018, "L'aventure") ;
   ```
3. ```
   INSERT INTO "oeuvres" VALUES (NULL,
   "L'aventure, l'ennui, le sérieux", "Jankélévitch", "Vladimir", "Français", 1963,
   "Essai philosophique", 2018, "L'aventure") ;
   ```
4. ```
   ALTER TABLE "Oeuvres" ADD COLUMN "Commentaire" TEXT ;
   ```
5. ```
   UPDATE "Oeuvres" SET Commentaire =
   "Trois manières différentes de disposer du temps"
   WHERE Nom_Auteur = "Jankélévitch" ;
   ```
6. ```
   DELETE FROM "Oeuvres" WHERE Nom_Auteur = "Jankélévitch"
   AND Prenom_Auteur = "Vladimir" ;
   ```
7. ```
   SELECT * FROM Oeuvres ;
   ```
8. ```
   SELECT DISTINCT Nom_Auteur FROM Oeuvres ;
   ```
9. ```
   SELECT Nom_Auteur, titre, annee_publication FROM Oeuvres
   WHERE annee_publication >= (
   SELECT annee_publication FROM Oeuvres WHERE titre = 'Lettres persanes')
   ORDER BY annee_publication ;
   ```
10. ```
    SELECT COUNT(DISTINCT Titre) FROM Oeuvres WHERE Nom_auteur = 'Molière' ;
    ```
11. ```
    SELECT COUNT(*) FROM Oeuvres WHERE Prenom_auteur IS NULL ;
    ```
12. ```
    SELECT AVG(annee_publication) FROM Oeuvres WHERE nationalite_auteur = 'Français' ;
    ```
13. ```
    SELECT COUNT(Titre), COUNT(DISTINCT Nom_auteur) FROM Oeuvres ;
    ```
14. ```
    SELECT titre, nom_auteur FROM Oeuvres WHERE annee_publication =
    (SELECT MIN(annee_publication) FROM Oeuvres) ;
    ```
15. ```
    SELECT Nationalite_Auteur, COUNT(*) AS c FROM oeuvres
    GROUP BY Nationalite_Auteur ORDER BY c DESC;
    ```

10.1.1 Requêtes complexes sur la base de données de mes classes

Remarque 3. *Ces requêtes ont été testés avec le logiciel* `Base` *de la suite logicielle* `LibreOffice`*, voilà pourquoi les noms des tables et des colonnes sont indiquées entre guillemets, c'est une particularité propre à ce logiciel pour la syntaxe* `SQL` .

Utilisation des **jointures**

1. ```
   SELECT "nom_eleve" AS "Nom", "prenom_eleve" As "Prénom",
    "filiere_classe" AS "Filière", "identifiant_classe" AS "Classe"
    FROM "CLASSES" JOIN "ELEVES" ON "classe_eleve" = "id_classe"
    WHERE ("filiere_classe" = 'PC' AND "identifiant_classe" = 'e') ;
   ```
   ou
   ```
   SELECT "nom_eleve" AS "Nom", "prenom_eleve" As "Prénom",
    "filiere_classe" AS "Filière", "identifiant_classe" AS "Classe"
    FROM "CLASSES", "ELEVES"
    WHERE ("classe_eleve" = "id_classe" AND "filiere_classe" = 'PC' AND
          "identifiant_classe" = 'e') ;
   ```

2. ```
   SELECT "nom_eleve" AS "Nom", "prenom_eleve" As "Prénom",
    "filiere_classe" AS "Filière", "identifiant_classe" AS "Classe",
    "nom_devoir" AS "DS", "valeur_note" AS "Note"
    FROM "CLASSES", "ELEVES", "DEVOIRS", "NOTES"
    WHERE ("classe_eleve" = "id_classe" AND "classe_devoir" = "id_classe" AND
          "devoir_note" = "id_devoir" AND "eleve_note" = "id_eleve" AND
          "nom_devoir" = 'SI_1' AND valeur_note" > 10) ;
   ```

   Utilisation des **agrégations**

3. ```
   SELECT "devoir_note" AS "DS", AVG("valeur_note") AS "Moyenne" FROM "NOTES"
    GROUP BY "devoir_note" ;
   ```

4. ```
   SELECT "devoir_note" AS "DS", AVG("valeur_note") AS "Moyenne"
   FROM "NOTES" GROUP BY "devoir_note" HAVING AVG("valeur_note") >= 11 ;
   ```

5. ```
   SELECT "filiere_classe" AS "Filière", "identifiant_classe" AS "Classe",
    "nom_discipline" AS "Discipline", "nom_devoir" AS "DS",
    AVG("valeur_note") AS "Moyenne"
    FROM "CLASSES", "DISCIPLINES", "DEVOIRS", "NOTES"
    WHERE ("id_discipline" = "discipline_devoir" AND
          "classe_devoir" = "id_classe" AND "devoir_note" = "id_devoir" AND
          "filiere_classe" = 'MPSI' AND "identifiant_classe" = '2')
    GROUP BY "filiere_classe", "identifiant_classe", "nom_discipline",
             "nom_devoir" ;
   ```

10.2 Application 2 : gestion pédagogique des classes préparatoires

10.2.1 Jointure à deux tables

1. ```
   SELECT Lycees.nom, Villes.nom FROM Lycees JOIN Villes
   ON Lycees.id_ville = Villes.id WHERE Villes.departement = 44 ;
   ```

2. ```
   SELECT L.nom FROM Lycees AS L JOIN Villes AS V
   ON L.id_ville = V.id WHERE population <= 250000 ;
   ```

3. ```
   SELECT COUNT(*) FROM Lycees, Villes WHERE Lycees.id_ville = Villes.id
   AND Villes.nom = 'Nantes' ;
   ```

4. ```
   SELECT V.departement FROM Villes AS V, Lycees AS L WHERE V.id = L.id_ville
   AND L.nom = 'Bergson' ;
   ```

10.2.2 Jointure à trois tables et plus

1. ```
   SELECT L.nom, F.nom FROM Lycees AS L, Filieres AS F, Rel_Lyc_Fil AS RLF
   WHERE L.id = RLF.id_lyc AND F.id = RLF.id_fil GROUP BY L.nom ;
   ```

2. ```
   SELECT L.nom FROM Lycees AS L, Filieres AS F, Rel_Lyc_Fil AS RLF
   WHERE L.id = RLF.id_lyc AND F.id = RLF.id_fil AND F.nom = 'PCSI' ;
   ```

3. ```
   SELECT F.nom FROM Lycees AS L, Filieres AS F, Rel_Lyc_Fil AS RLF, Villes AS V
   WHERE L.id = RLF.id_lyc AND F.id = RLF.id_fil
   AND L.nom = 'Clemenceau' AND V.nom = 'Nantes' ;
   ```

4. ```
   INSERT INTO Rel_Lyc_Fil(id_lyc, id_fil)
   VALUES ((SELECT L.id FROM Lycees AS L, Villes AS V WHERE L.id_ville = V.id AND
    L.nom = 'Clemenceau' AND V.nom = 'Nantes'),(SELECT F.id FROM Filieres AS F
   WHERE F.nom = 'PTSI')) ;
   ```

5. ```
   DELETE FROM Rel_Lyc_Fil WHERE id_lyc = (SELECT L.id FROM Lycees AS L, Villes
   AS V WHERE L.id_ville = V.id AND L.nom = 'Clemenceau' AND V.nom = 'Nantes')
   AND id_fil = (SELECT F.id FROM Filieres AS F WHERE F.nom = 'PTSI') ;
   ```

6. ```
   SELECT F.nom, L.nom, V.nom FROM Lycees AS L, Filieres AS F, Villes AS V,
   Rel_Lyc_Fil AS RLF WHERE L.id = RLF.id_lyc AND F.id = RLF.id_fil
   AND L.id_ville = V.id AND V.departement = 72 GROUP BY F.nom;
   ```

7. SELECT V.nom FROM Villes AS V, Filieres AS F, Lycees AS L, Rel_Lyc_Fil AS RLF
   WHERE F.nom = 'PSI' AND F.id = RLF.id_fil AND RLF.id_lyc = L.id
   AND L.id_ville = V.id ;
8. SELECT F.nom FROM Filieres AS F, Rel_Lyc_Fil AS RLF, Lycees AS L, Villes AS V
   WHERE V.population <= 250000 AND V.id = L.id_ville AND L.id = RLF.id_lyc
   AND RLF.id_fil = F.id ;

10.3 Application 3 : Élevage de Pokemons

1. SELECT E.nom, P.nom FROM Pokemons AS E JOIN Pokemons AS P ON P.id = E.idpere
   WHERE E.idmere = (SELECT id FROM Pokemons WHERE nom = 'Ludicolo') ;
2. SELECT E.nom, PE.nom FROM Pokemons AS E JOIN Pokemons AS PE
   ON (PE.idmere = E.id OR PE.idpere = E.id)
   WHERE E.idmere = (SELECT id FROM Pokemons WHERE nom = 'Ludicolo') ;
3. SELECT F1.nom FROM Pokemons AS F1 JOIN Pokemons AS F2
   ON (F1.idpere = F2.idpere AND F1.idmere = F2.idmere)
   WHERE (F1.sexe = 'M' AND F2.sexe = 'M') ;
4. SELECT COUNT(*) FROM Pokemons AS E JOIN Pokemons AS P
   ON E.idpere = P.id JOIN Pokemons AS M ON E.idmere = M.id
   WHERE E.annee_mort <= P.annee_mort AND E.annee_mort <= M.annee_mort ;