Bases du SQL

1 Présentation de la base de données

Nous disposons d'une base de données comprenant 3 tables :

- la table **eleves** comprenant les champs
  - id (clé primaire, c'est l'identifiant d'un élève)
  - nom
  - prenom
  - groupe_colle
- la table **planning_colle** comprenant les champs
  - id (clé primaire)
  - semaine
  - groupe
  - code_colle
- la table **colles_desc**
  - nom
  - code (clé primaire)
  - jour
  - debut
  - fin
  - matiere

On va interagir avec cette base via python, en utilisant du code non montré ici. Il faut juste savoir que nous allons directement faire exécuter les requêtes et afficher les résultats.

2 Projection

Dès que l'on veut obtenir des données depuis la base, on utilise la syntaxe de base suivante :

`SELECT une ou des colonnes FROM une table`

L'opération de ne selectionner que certaines colonnes est parfois appelée projection.

```
    affiche(req("SELECT nom, prenom, groupe_colle FROM eleves"))
```

```
nom             prenom          groupe_colle
Ait             Anthony         1
Ameye           Adrien          2
Avril           Emmanuel        2
Bailleux        Fabrice         1
Barrandon       Enguerrand      2
Begag           Faycal          3
Calbry          Benjamin        3
Chanellière     Lucas           4
Chaufournais    Antoine         4
Crapanne        Antoine         4
Deneux          Antonin         5
Durand          Pierre          6
Eveaert         Flavien         5
Gassama         Ibourahima      5
Genin           Boris           6
Hue             Louis           7
Kaim            Said            6
Khrajac         Louis           8
Lahmar          Jordan          7
Laine           Quentin         8
Le Bourhis      Pierre          8
Loyer           Alexis          9
Malitourne      Charles         9
Orvoën          Eugénie         7
Paumelle        Adelaïde        9
Poth            Léo             10
Rainglet        Benoît          11
Romain          Baptiste        10
Rouville        Alban           12
Roux            Etienne         11
Salf-Bommet     Julia           12
Vadcard         Thibaut         11
```

3 Filter les données

Une première opération courante est de vouloir filter les lignes retournées : on ne souhaite obtenir que celles qui vérifient une condition. La syntaxe est

```
SELECT ... FROM ... WHERE condition
```

Pour écrire une condition multiple, on peut utiliser les mots clés **OR** ou **AND**

```
affiche(req("""SELECT nom, prenom, groupe_colle
FROM eleves
WHERE groupe_colle >=5 AND groupe_colle <=10"""))
```

```
nom             prenom          groupe_colle
```

```
Deneux          Antonin         5
Durand          Pierre          6
Eveaert         Flavien         5
Gassama         Ibourahima      5
Genin           Boris           6
Hue             Louis           7
Kaim            Said            6
Khrajac         Louis           8
Lahmar          Jordan          7
Laine           Quentin         8
Le Bourhis      Pierre          8
Loyer           Alexis          9
Malitourne      Charles         9
Orvoën          Eugénie         7
Paumelle        Adelaïde        9
Poth            Léo             10
Romain          Baptiste        10
```

4 Ordonner les données

Pour trier les données de la table reçue suivant une ou des colonnes, on utilise encore un autre mot clé (on dit qu'on ajoute une clause)

`SELECT ... FROM ... [WHERE ...] ORDER BY une ou des colonnes [DESC]` Les crochets représentent des parties optionnelles de la requête. En particulier, le mot clé **DESC** indique que l'on veut un tri par ordre décroissant.

```
    affiche(req("""SELECT nom, prenom, groupe_colle
    FROM eleves
    WHERE groupe_colle >=5 AND groupe_colle <=10
    ORDER BY groupe_colle DESC, nom"""))
    # quand on indique plusieurs colonnes, le tri se fait d'abord suivant la␣
↪première
    # puis en cas d'égalité suivant la deuxième colonne indiquée.
```

```
nom             prenom          groupe_colle
Poth            Léo             10
Romain          Baptiste        10
Loyer           Alexis          9
Malitourne      Charles         9
Paumelle        Adelaïde        9
Khrajac         Louis           8
Laine           Quentin         8
Le Bourhis      Pierre          8
Hue             Louis           7
Lahmar          Jordan          7
Orvoën          Eugénie         7
```

```
Durand          Pierre          6
Genin           Boris           6
Kaim            Said            6
Deneux          Antonin         5
Eveaert         Flavien         5
Gassama         Ibourahima      5
```

5 Utiliser plusieurs tables

Il arrive souvent que les données voulues soient présente dans plusieurs tables. Dans ce cas on effectue une "jointure" des tables voulues grâce à la syntaxe

```
SELECT ... FROM table1     JOIN table2     [JOIN table3 ...]
```

Sous cette forme, la jointure effectue en réalité le produit cartésien des tables données.

Par exemple, si on souhaite connaître les colles pour chaque élève pour la semaine 4

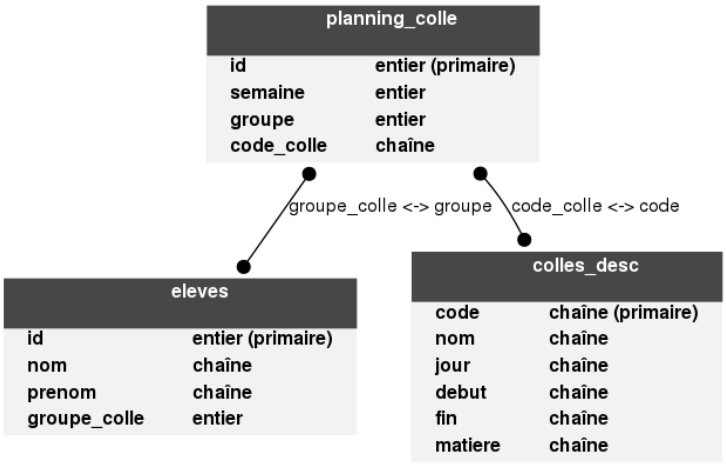
```
    affiche(req("""SELECT nom, prenom, code_colle
    FROM eleves
    JOIN planning_colle
    WHERE semaine = 4
    """)[:20])  # seulement les 20 premières lignes
```

```
nom     prenom  code_colle
Ait     Anthony AC1
Ait     Anthony AC2
Ait     Anthony AP
Ait     Anthony AT1
Ait     Anthony AT2
Ait     Anthony AVL
Ait     Anthony MB
Ait     Anthony MD
Ait     Anthony ML1
Ait     Anthony ML2
Ait     Anthony ML6
Ait     Anthony MR
Ait     Anthony PB
Ait     Anthony PC3
Ait     Anthony PC4
Ait     Anthony PC5
Ait     Anthony PD
Ait     Anthony PT
Ait     Anthony TL1
```

On a un problème en terme d'interprétation des données. Notre élève ne va sûrement pas pouvoir se dédoubler pour faire toutes ces colles... Dans le cas d'une jointure simple, chaque ligne de la table élève est associée à chaque ligne de la table planning_colle, ce qui n'est pas raisonable.

Méthode : avant d'écrire une jointure il faut identifier dans les tables concernées les colonnes qui contiennent des valeurs en commun. Dans ce cadre, les clé primaires prennent toutes leur utilité.

Dans notre exemple, on peut résumer les liens par le shéma suivant :



La syntaxe pour signifier ces lien est `SELECT ... FROM table1 JOIN table2 ON condition [JOIN table3 ON ...]`

```
    affiche(req("""SELECT nom, prenom, code_colle, groupe_colle
    FROM eleves
    JOIN planning_colle ON groupe = groupe_colle
    WHERE semaine = 4
    ORDER BY groupe_colle
    """)[:20])  # seulement les 20 premières lignes
```

nom	prenom	code_colle	groupe_colle
Ait	Anthony	AT1	1
Bailleux	Fabrice	AT1	1
Ait	Anthony	PC3	1
Bailleux	Fabrice	PC3	1
Ameye	Adrien	TL3	2
Avril	Emmanuel	TL3	2
Barrandon	Enguerrand	TL3	2
Ameye	Adrien	PC4	2
Avril	Emmanuel	PC4	2
Barrandon	Enguerrand	PC4	2
Begag	Faycal	AP	3
Calbry	Benjamin	AP	3
Begag	Faycal	ML2	3
Calbry	Benjamin	ML2	3
Chanellière	Lucas	TP3	4
Chaufournais	Antoine	TP3	4
Crapanne	Antoine	TP3	4
Chanellière	Lucas	ML1	4
Chaufournais	Antoine	ML1	4

Cette fois seules les lignes vérifiant la condition de jointure sont retournées : on a associé seulement les colles correspondant au groupe de chaque élève et non plus toutes les colles pour chaque élève

```
len(req("""SELECT nom, prenom, code_colle, groupe_colle
FROM eleves
JOIN planning_colle ON groupe = groupe_colle
WHERE semaine = 4
ORDER BY groupe_colle
"""))
```

```
65
```

```
# sans la condition de jointure
len(req("""SELECT nom, prenom, code_colle, groupe_colle
FROM eleves
JOIN planning_colle
WHERE semaine = 4
ORDER BY groupe_colle
"""))
```

```
769
```

6 Préciser les notations : qualifier les noms et alias

Si on voulait, dans la requête précédente obtenir l'id des élèves en plus, on se trouverait devant une ambiguité.

```
req("""SELECT id
FROM eleves
    JOIN planning_colle ON groupe_colle = groupe""")
```

```
␣
↪---------------------------------------------------------------------------

OperationalError                          Traceback (most recent call␣
↪last)

........


OperationalError: ambiguous column name: id
```

Le gestionnaire de base de données ne peut pas savoir si on fait référence à la colonne id de la table eleves ou à celle de la table planning_colle.

Pour lever cette ambiguité, on utilise des noms qualifiés. Pour faire référence à une colonne, on peut utiliser la syntaxe `table.colonne`

```
len(req("""SELECT eleves.id
FROM eleves
    JOIN planning_colle ON groupe_colle = groupe"""))
# on calcule juste le nombre de lignes.
```

```
1921
```

De plus, chaque colonne et table peut se voir attribuer un *alias*, c'est à dire un nom de notre convenance qui fera référence à l'objet en question.

`object AS nom_choisi`

```
affiche(req("""SELECT nom, prenom, groupe_colle AS G
FROM eleves"""))
# ici la dernière colonne est renommé G
```

nom	prenom	G
Ait	Anthony	1
Ameye	Adrien	2
Avril	Emmanuel	2
Bailleux	Fabrice	1
Barrandon	Enguerrand	2
Begag	Faycal	3
Calbry	Benjamin	3
Chanellière	Lucas	4
Chaufournais	Antoine	4
Crapanne	Antoine	4
Deneux	Antonin	5
Durand	Pierre	6
Eveaert	Flavien	5
Gassama	Ibourahima	5
Genin	Boris	6
Hue	Louis	7
Kaim	Said	6
Khrajac	Louis	8
Lahmar	Jordan	7
Laine	Quentin	8
Le Bourhis	Pierre	8
Loyer	Alexis	9
Malitourne	Charles	9
Orvoën	Eugénie	7
Paumelle	Adelaïde	9
Poth	Léo	10
Rainglet	Benoît	11
Romain	Baptiste	10
Rouville	Alban	12
Roux	Etienne	11
Salf-Bommet	Julia	12
Vadcard	Thibaut	11

```
    affiche(req("""SELECT el.id, nom, prenom, code_colle
    FROM eleves as el
    JOIN planning_colle ON groupe = groupe_colle
    WHERE semaine = 5
    """)[:10])
    # un alias de table permet d'alléger les notations qualifiées.
```

id	nom	prenom	code_colle
1	Ait	Anthony	TP2
4	Bailleux	Fabrice	TP2
1	Ait	Anthony	MD
4	Bailleux	Fabrice	MD
2	Ameye	Adrien	AT1
3	Avril	Emmanuel	AT1
5	Barrandon	Enguerrand	AT1
2	Ameye	Adrien	MB
3	Avril	Emmanuel	MB

7 Calculer de nouvelles données

Il arrive que l'on souhaite obtenir plutôt le nombre de lignes correspondant à un critère, ou obtenir la moyenne d'une quantité.

Les outils correspondants en SQL s'appellent *fonction d'aggrégation*. Il y en a plusieurs - count : compte le nombre de lignes - min, max : trouve le minimum ou le maximum - avg : calcule la moyenne - sum : calcule la somme

```
    # l'étoile * représente toutes les colonnes
    affiche(req("""SELECT count(*)
    FROM eleves
    """))
    # donne le nombre d'enregistrement dans la table élèves
```

count(*)
32

```
    affiche(req("""SELECT count(*)
    FROM eleves
    WHERE nom>='M'
    """))
    # nombre d'élève dont le nom de famille est après "M" dans l'ordre
    # alphabétique
```

count(*)
10

Il arrive souvent que l'on veuille obtenir plusieurs données calculées. Par exemple, on pourrait vouloir obtenir le nombre d'élève **par** groupe de colle (le mot clé dans cette requête est "par").

Dans ce cas on regroupe les données en utilisant une clause **GROUP BY**

`SELECT ... FROM ... [JOIN ...] [WHERE ...] GROUP BY colonne`

Dans ce cas, les lignes retournées par le début de la requête (select from) sont regroupées suivant la valeur contenue dans la colonne indiquée par la clause **GROUP BY** : on obtient une ligne par valeur différente contenue dans cette colonne.

```
    affiche(req("""SELECT groupe_colle, count(*) AS nb_eleves
    FROM eleves
    GROUP BY groupe_colle"""))
```

groupe_colle	nb_eleves
1	2
2	3
3	2
4	3
5	3
6	3
7	3
8	3
9	3
10	2
11	3
12	2