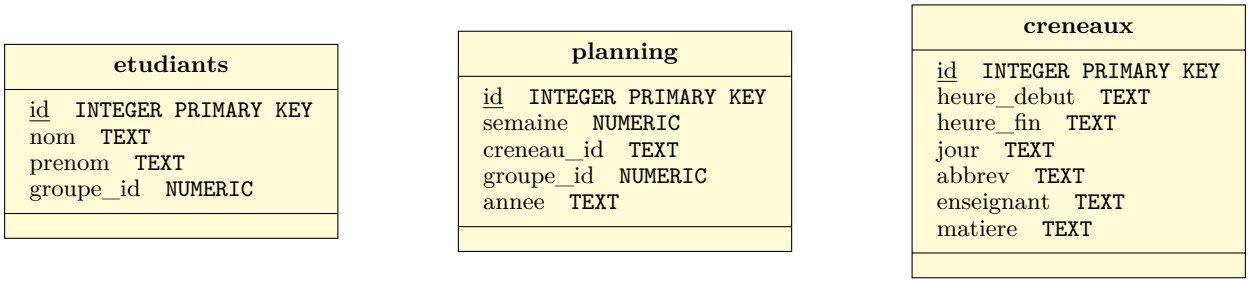
Préliminaires

Ouvrez Google Chrome puis installer l'extension SQLite Manager. Une fois cette extension installée, vous pouvez directement glisser-déposer la base de données `scope.sqlite` dans l'onglet correspondant puis les requêtes se tapent directement dans le cadre jaune contenant l'aide.
Pour passer à la ligne utiliser SHIFT + Entrée et utiliser Entrée pour exécuter la requête voulue.

Première partie

Base de données du colloscope.



Créez un fichier texte ou sauvegarder les requêtes qui fonctionnent, pour référence future.

Requêtes sur une seule table

1. Donner les nom et prénoms de tous les élèves.
2. Donner les nom et prénoms de tous les élèves classés par ordre alphabétique des prénoms.
3. Trouver le numéro de votre groupe de colle (avec ou sans requête), puis les élèves qui ont le même groupe.
4. Donner tous les créneaux de colle de mathématiques (on pourra utiliser `SELECT *` qui retourne toutes les colonnes) qui se terminent avant 16H (inclus).
5. La *fonction* `count` permet de compter le nombre de lignes. Etant une donnée que l'on souhaite récupérer, elle trouve sa place dans la clause `SELECT` sous la forme `count(*)`.
   Donner le nombre de créneaux de colle se déroulant le jeudi. On pourra d'abord s'interroger sur les valeurs dans la colonne `jour`
6. Donner le nombre de créneaux qui commencent à 17H ou plus tard, ou finissent avant midi (inclus).
7. Trouver toutes les valeurs du champ `annee` dans la table `planning` (on pourra utiliser `SELECT DISTINCT` à la place de `SELECT`). Pour chaque valeur trouvée, effectuer la requête qui compte le nombre de ligne pour cette année en question. Comparer à

   ```
   SELECT count(*), annee
   FROM planning
   GROUP BY annee
   ```
8. En adaptant la requête précédente, trouver le nombre de créneaux commençant après 17H (inclus) pour chaque matière.

Requêtes sur plusieurs tables

9. Trouvez les éventuels champs en commun dans les différentes tables, ainsi que le ou les champs faisant référence à une autre table. On doit pouvoir lier `planning` aux deux autres tables.
10. Donner le nom et le prénom des élèves ayant colle en semaine 15 de l'année 20-21.
11. Donner une table avec nom des colleurs et les semaines pour le groupe 17 (ou le groupe d'id qui vous convient), en triant par semaines croissantes.
12. Donner le nombre de colle ayant lieu le lundi pour votre groupe.
13. Donner une table avec nom de l'élève, le colleur et la matière pour la semaine 25. de l'année 17-18.
    Cette fois il nous faut des informations provenant des 3 tables. Il est possible d'utiliser plusieurs `JOIN` dans une même clause `FROM`

Pour aller plus loin

14. Pour alléger les notations `table.colonne` en cas de jointure, ou pour clarifier les noms de colonnes calculées, on peut utiliser des *alias*. Chaque nom de colonne de la clause `SELECT` et chaque table déclarée dans la clause `FROM` (y compris après un `JOIN`) peut s'écrire

`nom as alias`

ou même

`nom alias`

Par exemple on peut écrire

```
SELECT count(*) nb
FROM eleves
```

et la colonnes retournée est maintenant nommée `nb`. On peut évidement utiliser ce nom autre part dans la requête.
En ce qui concerne le nom des tables, on procède comme suit :

```
SELECT e.groupe, e.nom, p.semaine
FROM eleves e
    JOIN planning p ON ...
WHERE e.annee = ...
```

En utilisant un alias pour la colonne calculée, donner le numéro des groupes ainsi que le nombre d'élève par groupe.

15. Il existe d'autres fonctions que `count` qui sont souvent utiles : `avg, min, max, sum`. Ces fonctions prennent comme argument une colonne (celle dont on veut calculer la moyenne, le min, le max ou la somme).
Trouver le numéro de la première semaine où le groupe 15 a eu colle avec "M. Louatron".

16. Quand on veut filtrer les lignes de la table attendue selon une condition qui porte sur une donnée calculée (ie. non présente directement dans une table), on ne peut pas placer cette condition dans une clause `WHERE`, mais on doit la placer dans une clause spéciale : `HAVING`.
Par exemple, pour trouver les colleurs qui ont fait plus de 100 colles, on commence par compter le nombre de ligne de `planning` en groupant par enseignant, puis on filtre :

```
SELECT count(*) nb, c.enseignant
FROM planning p
    JOIN creneaux c ON p.creneau_id = c.id
GROUP BY c.enseignant
HAVING nb >= 100
```

Trouver tout d'abord le nombre de colle effectué par groupe de colle. Trouver ensuite quels groupes ont fait au moins 4 colles avec "M. Courteaud".

17. Trouver les noms et prénoms des 5/2 présents dans la table `eleves`. Indication : pour éviter les faux positifs, on pourra donner deux colonnes dans la clause `GROUP BY`. De plus, les heureux élus sont les seuls à être présents 3 fois dans cette table (ils ont un groupe par année de présence).

18. Gardons en tête que le résultat d'un requête est une table (comme les autres, pas de discrimination en SQL). Il est tout à fait possible de placer une table calculée par une requête dans une clause `FROM`, à condition de la mettre entre parenthèse.
Donner le nombre moyen d'élève par groupe.

19. Trouver les élèves dans des groupes de 2.

20. Trouver les paires d'élèves ayant le même nom de famille. Ici on veut comparer des données provenant deux fois de la même table. On pourra faire une "auto-jointure", c'est à dire joindre une table avec elle même

Deuxième partie
Méthodologie pour rédiger une requête sur une BDD

```
SELECT [DISTINCT]..., [COUNT/SUM...]
FROM ... [JOIN .. ON ..]
WHERE...
[GROUP BY... HAVING ...]
[ORDER BY...]
```

tests `AND, OR, IN` (*sous-requête*) *qui peuvent s'enchaîner*, `NOT, <,>, =...`
fonctions `COUNT(), SUM(), MAX(), MIN(), AVG()`
joker *, typiquement `SELECT *`, `COUNT(*)`.
Remarque : on peut également utiliser `COUNT(DISTINCT une_colonne)`.
Nous proposons une méthodologie simple pour répondre à une requête basique exprimée « en français ».
Comment remplir une requête de type `SELECT ...FROM ...WHERE...` ?

1. **Construction de la clause** `SELECT`

   (a) Identifier les champs (= attributs) qui doivent être affichés, et les tables les contenant.

(b) Mettre ces tables dans la clause `FROM` et éventuellement donner un nom de variable à chaque table (un alias)

(c) Mettre les champs à afficher, si nécessaire préfixés par le nom de variable de la table correspondante dans la clause `SELECT`

(d) (si nécessaire) Identifier les fonctions (`SUM`, `COUNT`...) à calculer, et les attributs nécessaires au calcul de ces fonctions, puis si nécessaire, ajouter des tables supplémentaires dans la clause `FROM`, et enfin ajouter la fonction dans la clause `SELECT`

2. **Construction des jointures** Rajouter la/les jointures avec `JOIN` et les *conditions de jointure* après `ON`, permettant de lier toutes les tables de la clause `FROM` les unes aux autres. Une telle jointure se fait en utilisant les *clés étrangères* des tables en question. Il peut arriver qu'il ne soit pas possible de lier une table aux autres. Cela signifie qu'il faudra introduire une ou plusieurs autres tables intermédiaires permettant de lier cette table par le biais d'un chemin composé de clés étrangères.
3. **Construction des groupes** (optionnel)

   (a) Identifier les champs utilisés pour le partitionnement en groupes et les mettre dans la clause `GROUP BY`.

   (b) Compléter si nécessaire les tables de la clause `FROM`.

   (c) Compléter la clause `SELECT` en rajoutant les champs utilisés pour le partitionnement. En général, ces champs auront déjà été identifiés car ils vont a priori apparaître dans le résultat final, puisqu'ils s'agissent de critères de regroupement.

   EXEMPLE une requête qui calcule le revenu moyen en fonction de la ville affichera forcément le nom de la ville de chaque groupe.
4. **Construction des restrictions (`WHERE` et `HAVING`)**

   (a) Regarder sur quels champs (ou groupes de champs) de quelles tables portent les restrictions.

   (b) Rajouter ces tables, si elles ne sont pas encore présentes, dans la clause `FROM`.

   (c) Rajouter les restrictions sur les champs en question dans la clause `WHERE`.

   REMARQUE IMPORTANTE une condition de restriction peut tout à fait s'exprimer en utilisant une *sous-requête*.

   Par exemple :

   ```
   SELECT nom FROM tb_eleve WHERE villenaissance NOT IN (SELECT ville FROM tb_villes WHERE region <>
   'normandie')
   ```

   (d) Rajouter les restrictions qui portent sur des valeurs agrégées (`GROUP`) dans la clause `HAVING`.
5. **Construction des sous-requêtes** (optionnel)

   Pour chaque sous requête utilisée dans la clause `WHERE`, vérifier qu'elle a besoin ou non d'être paramétrée.

   Une sous requête non paramétrée signifie que sa valeur ne dépend pas du moment où elle est calculée c'est-à-dire une sous requête peut être utilisée pour calculer le salaire moyen. Une sous requête paramétrée signifie que sa valeur dépend de la ligne qui est en train d'être traitée i.e. le fait qu'un idproprietaire donné possède un compte de type Livret A est une sous requête paramétrée. Dans le cas d'une sous requête paramétrée, celle-ci devra avoir une condition de paramétrisation dans la clause `WHERE`. Cette condition de paramétrisation fera intervenir un identifiant de table de la requête extérieure. (Voir exemples plus bas).
6. Présenter le résultat suivant un certain **classement**, **ORDER BY** (optionnel)

`A noter` On peut effectuer plusieurs requêtes (en appliquant la méthodologie précédente) puis les regrouper (`UNION`) ou en faire une différence (`EXCEPT`).