# 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)