SQL – FICHE

vocabulaire général

- **attribut** : une colonne aka un type de données dans une table
- **les arguments** : 'birth', '1805'… : ce qu'on précise pour une clause
- **subselect** : un select dans un select ; s'écrit comme une requête select, mais entre '()', à l'intérieur de la première requête select

commandes : spécifient l'action à effectuer
'SELECT', 'UPDATE', 'DELETE', 'INSERT INTO', 'CREATE DATABASE', 'ALTER DATABASE', 'CREATE TABLE', 'ALTER TABLE', 'DROP TABLE', 'CREATE INDEX' ,'DROP INDEX'

- **SELECT** : lance une requête : quelles données afficher dans la base
- un attribut = une colonne

clauses : spécifient comment traiter une commande

- **FROM** [nom_de_table] : lieu où chercher (table)
- **WHERE** : clause de sélection : filtre les données en fonction de certains critères (opérateurs) : on trie les données en fonction d'opérateurs appliqués à certains attributs
- **ORDER BY** : réordonner les résultats de la requête en fonction d'un argument
  - ORDER BY attribut **ASC** : par défaut – ordonner les résultats dans l'ordre croissant
  - ORDER BY attribut **DESC** : ordonner les résultats dans l'ordre décroissant
- **LIMIT** [numeral] : limiter le nombre de résultats affichés (LIMIT 4 => limité à 4)
- **GROUP BY** :
  - en anglais : optional clause of the SELECT statement that combines rows into groups based on matching values in specified columns. One row is returned for each group.
  - en français : groupe les lignes sélectionnées en se basant sur la valeur de colonnes spécifiées pour chaque ligne et renvoie une seule ligne par groupe.
- **HAVING** : agit comme WHERE, mais permet de filtrer les opérations résultant de regroupement (permet de travailler avec des fonctions d'agrégation ≠ WHERE qui permet de travailler sur des données) ; HAVING doit toujours s'utiliser après GROUP BY
- **AS** : alias, clause qui permet de renommer un argument (pour travailler avec les résultats d'une fonction d'agrégation, par exemple)
- **DISTINCT** : permet de supprimer les doublons dans le résultat d'une requête ; s'applique à un argument

opérateurs : les opérateurs servent à comparer/traiter plusieurs arguments dans une clause

- **opérateurs arithmétiques** : + (add), – (substract), * (multiply), / (divide), % (modulo)

- **opérateurs de comparaison** : =, <, >, ≥, ≤, <> (différent de)

- **compound operators** (//python) : += (add equals), -= (substract equals), *= (multiply equals), /= (divide equals), %= (modulo equals), &= (bitwise and equals), ^-= (bitwise exclusive equals), |*= (bitwise or equals)

- **opérateurs logiques** : ALL, AND, ANY, BETWEEN, EXISTS, IN, IS, LIKE, NOT, OR, SOME
  - **IS** : opérateur boléen (teste si une clause est True=1 ou False=0). On utilise 'IS' que pour 'IS NULL' ou 'IS NOT NULL' ; sinon, utiliser 'EQUALS' ; **IS NOT** = inverse de IS
  - **LIKE** : True if the operand is equal to one of a list of expressions. Permet de rentrer des arguments de manière +souple que 'IS' ou '=' avec une chaîne de caractères, en utilisant '%'

- **BETWEEN** : utilisé avec '**AND**', permet de traiter des plages de caractères ; inclut les bornes
- **NOT, AND, OR** : non, et, ou, assez logiquement

- **opérateur de concaténation** : '||' : permet de lier 2 arguments ensembles dans les résultats

jointures : requêter sur 2+ tables = faire une jointure ; par défaut, une jointure retourne le **produit cartésien** des tables => il faut filtrer le **produit cartésien** en croisant la clé primaire d'une table avec une clé étrangère

- **JOIN – équijointure / inner join / jointure interne / jointure naturelle** :
  - **intersection de A+B** ; jointure de base, où clé d'une table = clé extérieure d'une autre table
  - **syntaxe** : différentes manières d'écrire, plus ou moins sexy :
    - avec INNER JOIN...ON (le mieux) :
      - SELECT [attributs] FROM table1 INNER JOIN table2 ON [clause de contrainte]
    - avec INNER JOIN … USING si les 2 attributs d'identifiant à matcher sur les 2 tables ont le même nom :
      - SELECT [attributs] FROM table1 INNER JOIN table2 USING(nom_identifiant)
    - avec WHERE :
      - SELECT [attributs des 2 tables à afficher] FROM table1 JOIN table2 WHERE [clause de contrainte]
    - avec NATURAL JOIN si les 2 attributs d'identifiant à matcher sur les 2 tables ont le même nom :
      - SELECT [attributs] FROM table1 NATURAL JOIN table2

- **LEFT JOIN – jointure gauche** :
  - **A + intersection de A+B** ; toutes les données de table1 avec quand c'est possible les données supplémentaires de la table2
  - **syntaxe** : SELECT [attributs] FROM table1 LEFT JOIN table2 [USING(nom_identifiant) / ON / WHERE]

- **RIGHT JOIN – jointure droite** :
  - **B + intersection de A+B** ; toutes les données de table2 avec quand c'est possible les données supplémentaires de table1
  - **syntaxe** : SELECT [attributs] FROM table1 RIGHT JOIN table2 [USING / ON / WHERE]
  - **pas implémentée en SQLite =>** dans SQLite, LEFT JOIN = miroir de RIGHT JOIN : FROM table1 LEFT JOIN table2 = FROM table2 RIGHT JOIN table1
  - **exemple** : table1=prénoms, table2=noms => retourner tous les noms de la table2 avec quand c'est possible, les prénoms de la table1

- **FULL JOIN – jointure totale** :
  - **A+B sans doublons** ; combiner les résultats des 2 tables, les associer entre eux grâce à une condition et remplir avec des valeurs NULL si la condition n'est pas respectée
  - **syntaxe** : SELECT [attributs] FROM table1 FULL JOIN table2 ON table1.id = table2.id
  - **pas implémentée dans SQLite** => il faut joindre 2 LEFT JOIN en miroir avec UNION : table1 LEFT JOIN table2 UNION table2 LEFT JOIN table1

fonctions d'aggrégation

permettent d'exprimer des conditions sur des groupes de lignes et de constituer le résultat :

- 'COUNT()' – compte le nombre de lignes
- 'AVG()' – calcule la moyenne de l'expression
- 'MIN()' – calcule la valeur minimale de l'expression
- 'MAX()' calcule la valeur maximale de l'expression
- 'SUM()' – calcule l'addition de la totalité des résultats

les wildcards :

***** : un ou plusieurs caractères inconnus

? : un seul caractère inconnu

[] : représente l'un des caractères parmi tous ceux qui sont indiqués entre crochets. *Exemple : [aA] permet de rendre le « a » case insensitive*

! : exclut les caractères spécifiés entre crochets. *Exemple : [!oa] excluera 'o' et 'a'*

- : représente un caract parmi une plage de caractères entre crochets. *Exemple [a-t]*

: représente un chiffre. *Exemple 2#5* retourne tous les nombres entre *205 et 295*

% (ne fonctionne qu'avec 'LIKE') : représente un ou plusieurs caractères inconnus.