TP n° 6 : corrigé.

MP Lycée Camille Jullian

12-19 mars 2022

Bases de Données.

1. SELECT LastName, FirstName FROM customers
2. SELECT Name FROM artists WHERE Name LIKE '%Z%'
   (rappel : LIKE permet de faire des recherches de chaînes de caractères incomplètes, en utilisant le % pour désigner n'importe quelle sous-chaine de caractère, et _ pour désigner n'importe quel caractère, ce qui permet par exemple de chercher tous les noms de musiciens de cinq lettres commençant par L avec un LIKE 'L____')
3. Un exemple classique de recherche croisée nécessitant une jointure. On peut pour cela utiliser un table1 JOIN table2 ON condition ou plus simplement un NATURAL JOIN sans le ON si les deux attributs permettant l'identification portent le même nom dans les deux tables (et sont les seuls dans ce cas !).
   SELECT Title FROM albums NATURAL JOIN artists ON Name='Iron Maiden'
4. C'est plus pénible en effet, d'une part parce qu'il faut faire deux jointures pour relier entre elles trois tables (le lien entre tracks et artists passe nécessairement par albums), mais surtout car on ne peut pas faire deux NATURAL JOIN successifs dans la mesure où tracks et artists possèdent un attribut commun Name qui ne représente pas la même chose et va perturber la recherche.
   SELECT tracks.Name FROM tracks NATURAL JOIN albums JOIN artists
   ON albums.Artistid=artists.Artistid WHERE artists.Name='Iron Maiden'
5. SELECT Title FROM albums NATURAL JOIN tracks JOIN genres ON tracks.GenreId=genres.GenreId WHERE genres.Name='Jazz' AND Milliseconds>600000
6. SELECT COUNT(*) FROM playlists
   On peut mettre n'importe quoi à l'intérieur du COUNT, le résultat affiché sera le même (on compte de toute façon le nombre de lignes affichées).
7. 10 playlists suffiront, on est ici obligés d'utiliser une option GROUP BY pour regrouper les chansons par playlists avant de les compter, et en plus de ça d'afficher les résultats par ordre décroissant (option ORDER BY attribut DESC) et avec une limite de 10 résultats (LIMIT 10).
   SELECT playlists.Playlistid, COUNT(TrackId) AS c FROM playlists NATURAL JOIN playlist_track GROUP BY playlist_track.Playlistid ORDER BY c DESC LIMIT 10
8. SELECT AVG(Milliseconds) FROM tracks WHERE Composer LIKE '%Steve Harris%'
9. SELECT LastName, FirstName FROM employees ORDER BY HireDate
10. SELECT * FROM invoices NATURAL JOIN customers WHERE FirstName='Camille' and LastName='Bernard' AND Total=(SELECT MAX(Total) FROM invoices NATURAL JOIN customers WHERE FirstName='Camille' AND LastName='Bernard')
    Oui, c'est très laid, mais la syntaxe « naturelle » avec par exemple un SELECT InvoiceId, MAX(total) ne fonctionne pas (ça affiche bien le total maximal, mais pas associé à la bonne facture !).

11. SELECT LastName, FirstName, SUM(total) FROM invoices NATURAL JOIN customers WHERE Country='Brazil' GROUP BY CustomerId
12. DELETE FROM customers WHERE Country='Russia'

    Note : l'exemple était très mauvais puisqu'il n'y a aucun client russe à l'origine dans la base, ne soyez donc pas surpris si la commande ne produit aucun résultat.
13. INSERT INTO customers VALUES (60,'Rou','Poil','Lycée Camille Jullian','42 rue des Mathématiques','Bordeaux',NULL,'France',38000,NULL,NULL,NULL,3)

    Les NULL correspondent simplement à des attributs non renseignés (les cases grises quand on affiche les tables avec SQLite Manager).

Analyse numérique.

Calcul approché d'intégrales.

Les trois programmes reprennent les formules de mon résumé de cours. L'intégrale qu'on demande de tester a pour valeur exacte 9. Voici les résultats de mes tests à l'aide de la méthode des rectangles (le temps est ici donné en millisecondes) :

n	temps	valeur
100	0	8.86545
1 000	1	8.98650
10 000	5	8.99865
100 000	57	8.999865
1 000 000	619	8.9999865
10 000 000	6 044	8.9999987

Sans surprise, le temps d'exécution est proportionnel à n (l'algorithme est censé être linéaire) et a précision obtenue inversement proportionnelle à n, ce qui nécessaite d'énormes valeurs de n pour obtenir une bonne approximation. Même tableau avec la méthode des trapèzes :

n	temps	valeur
10	0	9.045
100	0	9.00045
1 000	1	9.0000045
10 000	6	9.000000045
100 000	63	9.00000000045

Toujours un temps de calcul linéaire (et très proche de celui de la méthode des rectangles) mais une approximation nettement meilleure (on gagne deux décimales au lieu d'une à chaque multiplication par 10 du nombre de trapèzes). Enfin, pour la formule de Simpson, on va changer de calcul car cette méthode donne une valeur exacte pour l'intégrale d'une fonction de degré 2. Calculons plutôt $\int_0^3 x^5 \, dx$, qui doit donner la valeur 121.5 :

n	temps	valeur
10	0	121.50152
20	0	121.500095
50	0	121.50000243
100	0	121.500000152
200	0	121.500000009
500	1	121.5
1 000	3	121.5

C'est évidemment encore nettement plus efficace. Remarque en passant : les float(n) qui trainent dans mes programmes sont totalement facultatifs avec des versions récentes de Python, mais j'ai sur mon ordi personnel une version pas du tout à jour où les division d'entiers ne sont pas effectuées correctement.

Méthode d'Euler pour une équation du premier ordre.

Cf le programme écrit dans le fichier python. C'est exactement ce qu'on a fait ensemble en TP de toute façon. On peut bien sûr changer la valeur de n pour observer la qualité de la convergence des approximations vers la vraie courbe de l'exponentielle.

Résolution d'un système différentiel.

J'ai écrit un programme général à partir de deux équations modélisées par des fonctions à trois variables $F(x, y, t)$ et $G(x, y, t)$. Vous noterez dans mon programme que pour calculer la nouvelle valeur de la fonction2, j'utilise fonction1[-2] comme premier argument de la fonction G, pour ne pas utiliser la nouvelle valeur qu'on vient de calculer à la ligne précédente, mais bien celle correspondant à l'étape précédente. On peut éviter ce problème en mettant à jour les deux listes simultanément.

1. La fonction x correspond aux lapins et y aux renards. En effet, le facteur négatif devant le y dans l'expression donnant la valeur de x' montre que la population x croit d'autant moins vite que la population y est elevée (plus il y a de renards, plus les lapins se font bouffer) alors que c'est le contraire pour la population y (plus il y a de lapins, plus les renards ont de quoi bouffer).
2. Cf le fichier Python, on obtient des courbes quasiment périodiques (elles devraient l'être exactement).
3. On est alors sur une position d'équilibre, il n'y a aucune évolution. Si on prend un point de départ proche de cette situation d'équilibre, l'évolution sera modérée au fil du temps, elle sera de plus en plus violente si on s'éloigne nettement du point d'équilibre. Par exemple, avec $x_0 = 4.2$ et $y_0 = 1.4$, la population de lapins ne déborde pas de l'intervalle $[3.5, 4.5]$, alors qu'avec $x_0 = 12$ et $y_0 = 6$, on a un maximum qui approche 30 et un minimum extrêmement proche de 0.
4. Il suffit de remplacer dans mon programme les deux plot par un mpl.plot(fonction1,fonction2). On observe alors une espèce do'vale tournant périodiquement autour du point d'équilibre. Plus la situation initiale est proche du point d'équilibre, plus l'ovale est régulier (on est plus proche d'un triangle avec des situations initiales extrêmes).

Résolution d'une équation différentielle du second ordre.

On peut reprendre le programme précédent, en posant $x = y'$, et en écrivant les deux équations $y' = x$ et $x' = -\sin(y)$ (ou $-\sin(y) + k * y$ pour ajouter le frottement). Cf le fichier Python joint (on a intérêt quand on fait tourner le programme sur cette équation à ne tracer que la courbe de la fonction2, c'est-à-dire de y).