

Rapport TP4

Architecture Base de données

Etudiants :

BIBI thilleli
KHEMSINE Lydia

thillelibibi@gmail.com
lydiakhemsine@outlook.com

Structure de TP4 :

notre projet JAVA est structuré en 3 classe :

1) TablesandBlocks.java :

-create_relation

fonction qui prend en paramètre un entier nbr_lignesR et retourne un tableau de string contenant des couples de lettres de l'alphabet en faisant le produit cartésien de l'alphabet avec quelques lettres qui sont bien définies , puis on applique la fonction shuffle sur la liste afin de générer des couples aléatoires.

-make_block

fonction qui prend en paramètre une liste de String, 2 paramètres pour les numéros des blocs et retourne un tableau des éléments des blocks en faisant la concaténation en code ASCII des lettres.

-all_blocks

fonction qui appelle la fonction make_block pour construire les blocks et les met dans airtable en appelant la fonction block_to_airtable de la classe airTable.java.

2) air_table.java

-block_to_airtable

fonction qui utilise une requete http (POST) pour mettre les blocs dans airtable, le code est le suivant :

```
13 .version(HttpClient.VERSION_11_0_2);
14 .build();
15
16 public static void block_to_airtable(int[] block,String blockname) throws IOException, InterruptedException {
17
18     StringBuilder json = new StringBuilder() ;
19     json.append("{ \"records\": [");
20
21     for (int i=0;i < 9; i++) {
22         json.append(" { \"fields\": { \"Name\":\"" + block[i] + "\" },");
23     }
24     json.append(" { \"fields\": { \"Name\":\"" + block[9] + "\" }}");
25     json.append("]");
26
27     String jsonString = json.toString() ;
28
29     HttpResponse response;
30
31     // add json header
32     HttpRequest request = HttpRequest.newBuilder()
33         .POST(HttpRequest.BodyPublishers.ofString(jsonString))
34         .uri(URI.create(blockname))
35         .setHeader("User-Agent", "Java 11 HttpClient Bot") // add request header
36         .header("Content-Type", "application/json")
37         .build();
38     response = httpClient.send(request, HttpResponse.BodyHandlers.ofString());
39     System.out.println(response);
40     System.out.println(response.body());
41 }
42 }
```

ion Console

ation] /usr/lib/jvm/jdk-11.0.2/bin/java (18 nov. 2020 à 23:02:34 – 23:02:49)

Writtable Smart Insert 46:5:1400

-tabledescriptor :

fonction qui sert à mettre les descripteur dans airtable en utilisant la requête POST HTTP.

-GetTable :

fonction utilisée par la classe NestedLoop pour récupérer les blocs de airtable et faire la jointure en utilisant la requête GET de HTTP. Après on parse le résultat de la requête pour avoir le nom des blocks. Le code est le suivant :

```
72 }
73
74 public int[] GetTable(String nametable) throws IOException, InterruptedException {
75     HttpRequest request = HttpRequest.newBuilder()
76         .GET()
77         .uri(URI.create(nametable))
78         .build();
79
80     HttpResponse<String> reponse = httpClient.send(request, HttpResponse.BodyHandlers.ofString());
81     System.out.println(reponse.statusCode());
82     //System.out.println(reponse.body());
83
84
85     int[] tab = new int[10];
86     Pattern pattern = Pattern.compile("\\Name\\:\\.\\(\\.\\+?\\)\\.");
87     Matcher matcher = pattern.matcher(reponse.body());
88
89     int i = 0;
90     while (matcher.find()) {
91         //System.out.println(matcher.group(1));
92         tab[i] = Integer.valueOf(matcher.group(1));
93         i++;
94     }
95
96
97     return tab;
98 }
99 }
100
101
```

Application] /usr/lib/jvm/jdk-11.0.2/bin/java (18 nov. 2020 à 23:02:34 – 23:02:49)

Writable Smart Insert 96:8:3120

3) NestedLoop.java

-nestedloop :

fonction qui sert à faire la jointure entre les blocs de R et de S, le code de la jointure est le suivant :

```
42 while(lenr<Urir.length){
43     //charger un block
44     Blockr = airtable.GetTable(Urir[lenr]);
45     lens = 0;
46     while(lens<Uris.length) {
47         Blocks = airtable.GetTable(Uris[lens]);
48         //nestedloop
49         int i=0;int j ;
50         while(i<Blockr.length) {
51             j=0;
52             while(j<Blocks.length) {
53                 if (Blockr[i]==Blocks[j] && Blockr[i]!=0 ) {
54                     if (k1<10) {
55                         result1[k1] = Blockr[i];
56                         k1++;
57                     }
58                     else if (k2<10) {
59                         result2[k2] = Blockr[i];
60                         k2++;
61                     }
62                     else if (k3<10) {
63                         result3[k3] = Blockr[i];
64                         k3++;
65                     }
66                     else if (k4<10) {
67                         result4[k4] = Blockr[i];
68                         k4++;
69                     }
70                 }
71                 j++;
72             }
73             i++;
74             lens++;
75         }
76     }
77 }
```

Console

ation] /usr/lib/jvm/jdk-11.0.2/bin/java (18 nov. 2020 à 23:02:34 - 23:02:49)

Writable Smart Insert 69:31:2678

Les liens des bases :

<https://airtable.com/shrVJ85vp083bMJVv>

<https://airtable.com/shrmRPMtcVv0CnxE>

Exécution :

Faut exécuter la classe TableandBlocks et après exécuter la classe NestedLoop pour faire la jointure.

