Architectures des Systèmes de Bases de Données
TP4 Airtable Disk Nested Loop

Qian Christian | christian-qian@live.fr | Etudiant : 21964319

Le projet est composé de sept fichiers plus le fichier de test :

- un fichier Blockcreation pour la création des valeurs

- un fichier Inode représentant un descriptor et gérer ses blocks

- un fichier HttpClientDelete pour supprimer les valeurs d'une table (aussi avec l'aide d'une inode)

- un fichier HttpClientGet pour récupérer les valeurs d'un tableau

- un fichier HttpClientPost pour ajouter des valeurs sur une table (aussi avec l'aide d'un inode)

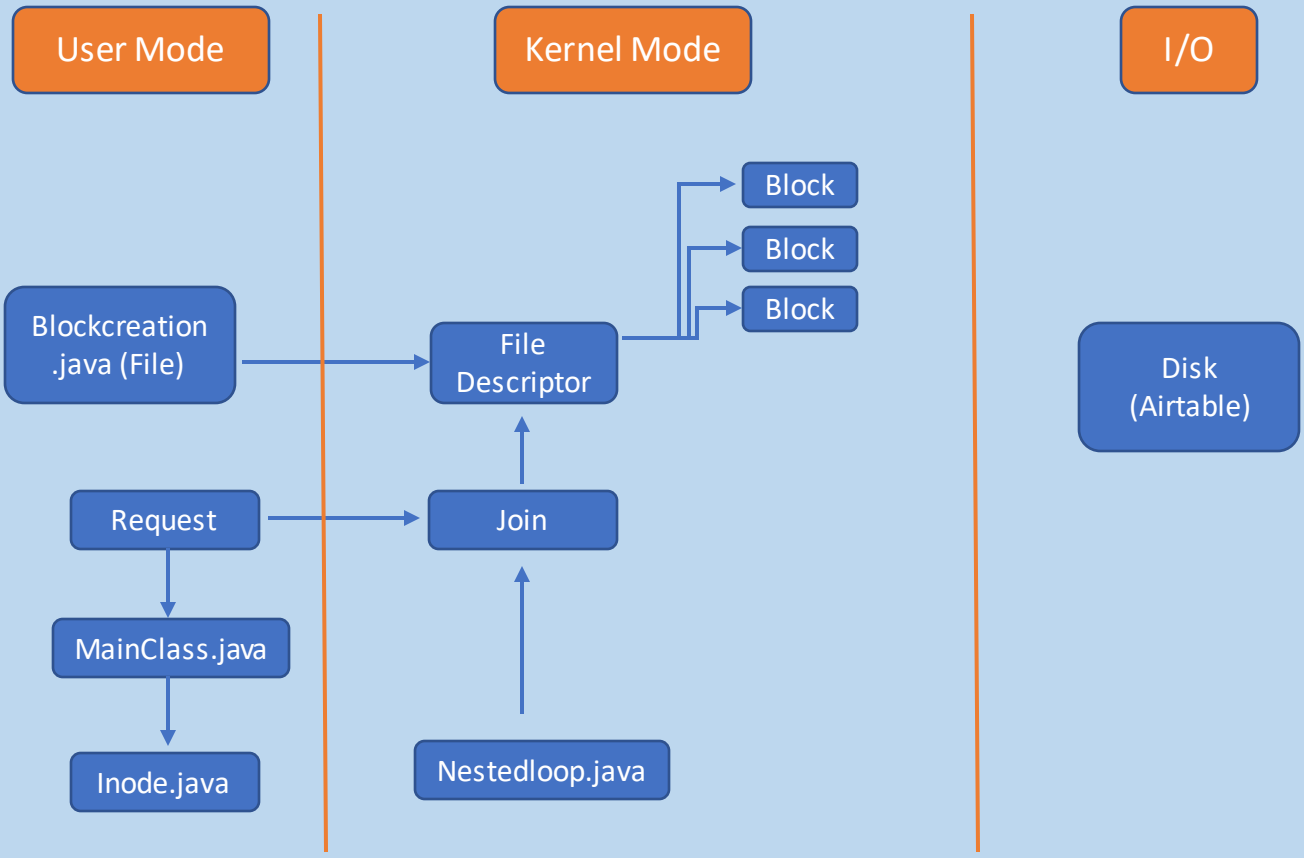
- un fichier Nestedloop pour effectuer un nested loop entre deux blocks , annonce s'il y a assez de valeur pour remplir complètement un block complet et l'ajoute au tableau courant

- un fichier MainClass avec trois fonctions, la première et seconde supprime les anciennes valeurs des inodes, blocks et ajoute les nouvelles, la seconde lance le Nestedloop entre R et S, puis écrit sur RS

Il est possible de changer le nombre de valeur de R et S; de changer la taille des inodes de R, S et RS; ainsi que changer la taille des blocks.

Il est bien sûr nécessaire de vérifier que les tables sur Airtable puissent contenir les valeurs de R, S et le résultat du Disk Nestedloop.

Diagramme UML



Algorithmes

Blockcreation.java

- newValues(int letter, int nbR, int nbS) : int[][]

Retourne un tableau avec deux array de taille nbR et nbS, la variable letter (entre 1 et 26) limite les valeurs de la deuxième liste d'alphabet pour la combinaison ASCII.

- inodeValues(int nbR, int nbS) : int[][]

Retourne un tableau avec trois array d'inode de taille nbR, nbS et min(nbR, nbS).

HttpClientDelete.java

-deleteWithInode(int [] inode) : void

Appelle deleteTable() et supprime tous les blocks de inode.

-deleteTable(String table) : void

Supprime toutes les valeurs d'une table.

HttpClientGet.java
-getTableValues(String table, int tabletaille) : int []
Recupere les valeurs d'une table de taille tabletaille.

-getTableId(String table) : List<String>
Retourne tous les ids d'une table (appelé seulement pour supprimer ces valeurs)

Inode.java
- addInodeValues(int[] values) : void
Supprimme et met à jour les pointeurs.

- deleteRelation() : void
Supprimme le contenu des blocks

- createRelation(int[] values) : void
Ajoute dans les blocks les valeurs values

- getBlock(int index) : int
Retourne la valeur du pointeur à la case i

- getBlockValues(int i) : int[]
Retourne les valeurs du block i

- addToBlock(int k, int[] values) : void
Ajoute values au block k

HttpClientPost.java
-postWithInode(int [] values, int [] inode) : void
Appelle postBlock() et ajoute values aux blocks à partir d'un inode.

-postBlock(String table, int [] values, int i, int j, boolean inode) : void
Ajoute les valeurs entre les indices i et j de values à un block ou un inode.

Nestedloop.java
-nloop(int[] r, int[] s, int[] before, boolean send, String table) : boolean
Effectue Nestedloop sur r , s et ajoute le résultat sur before.
S'il y a plus de résultat que l'array before peut stocker, on ajoute les valeurs de before à Airtable et on met à jour before avec les valeurs en trop et on renvoie true pour passer à la table suivante.
J'ai écrit de cette façon pour effectuer le moins d'appel Http que possible.

MainClass.java
-addInodeValues(int [][] v) : void
Supprime toutes les valeurs des descriptors S, R, RS et met à jour S, R, RS avec v.

-addNewValues(int [][] v) : void
Supprime toutes les valeurs des relations S, R, RS et ajoute les valeurs v à S et R.

- intToBlock(int i) : String
Convertit un int vers un nom de block

-exeNestedloop() : void
Parcours les blocks de R et S, appelle **Nestedloop.**nloop() pour ajouter les résultats sur Airtable.

Le Test

Pour mettre en place le test, j'ai effectué un Nestedloop simple sur les valeurs de **Blockcreation.**newValues() (les même valeurs de R et S ajoutés à Airtable) et stocké le résultat.
Après avoir effectué **MainClass.**exeNestedloop(), j'ai comparé le résultat entre le Nestedloop simple et le Disk Nestedloop, et on obtient bien un pass de JUnit.

L'exécution du test JUnit prend environ 25 secondes avec toutes les suppressions, recherches et envois de valeurs.

DATABASE LINK