SbornDB – documentazione

**Premessa**

In questo file sono racchiuse tutte le specifiche sul funzionamento del database, sui suoi costi in termini di velocità e di memoria.

Il progetto è diviso in diverse parti: gestione della memoria, strutture dati, corpo centrale e parser.

Di seguito ogni parte verrà analizzata nel dettaglio. Infine vi è una sezione dedicata all’analisi dei costi computazionali in ordine di tempo e spazio richiesto.

**Gestione della memoria**

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Leo integer malesuada nunc vel. Etiam erat velit scelerisque in dictum non consectetur. Tortor posuere ac ut consequat semper viverra nam libero justo. Dis parturient montes nascetur ridiculus mus mauris vitae. Arcu odio ut sem nulla pharetra diam. Vitae nunc sed velit dignissim sodales ut eu. Ornare aenean euismod elementum nisi quis. Mattis rhoncus urna neque viverra justo. Tortor at risus viverra adipiscing at in.

**Strutture dati**

In questa sezione sono descritte le varie strutture dati utilizzate e come sono collegate tra loro.

Le principali strutture dati utilizzate sono:

* Linked list
* Red Black Tree
* Heap (modificato)

La struttura delle Linked List non varia dalla regolare implementazione.

I Red Black Trees sono personalizzati per il progetto. Nella struttura RBTree è presente un puntatore alla radice dell’albero ``` struct RBTNode\* root; ``` e una chiave ``` int key; ```. La chiave può assumere due diversi significati: può indicare che sia un albero di tabelle o un albero di puntatori a record. Nel primo caso è indicato assumendo valore -2, mentre nel secondo caso assume valori tra 0 e MAX\_INT, che indicano l’indice della colonna di ogni tabella. Questa variabile non assume mai valore -1 in quanto è il valore ritornato dalla funzione “searchColumnIndex()” nel caso in cui la colonna richiesta non è stata trovata.

Per la implementazione dell’heap si rimanda alla precedente sezione “Gestione della Memoria”. (!!!—GIACK—!!! Va bene cosi o vuoi mettere la descrizione della struttura dati per la gestione della memoria qui anche te?)

*Il database è accessibile da un unico puntatore dichiarato globale.* È stata fatta questa scelta in quanto si assume di lavorare sempre sullo stesso database, seguendo le linee guida del progetto. In questo modo si evita di passare il puntatore al database in ogni funzione alleggerendo lo stack di sistema.

La struttura Database è un Red Black Tree nei quali nodi sono salvate le tabelle caricate in memoria.

La struttura RBTNode contiene i puntatori di una normale implementazione di un nodo di un red black tree: colore, testa dell’albero, padre, figlio destro e figlio sinistro. Inoltre contiene il puntatore ```void \* nodeValue;```, il quale contiene o la tabella o il puntatore al record a cui si riferisce il nodo stesso (come riportato precedentemente è possibile capire il tipo del puntatore nodeValue a seconda della chiave nella testa dell’albero).

La struttura tabella (```struct TableDB```) contiene: il nome della tabella (“name”), i nomi delle colonne (“columns”), il numero delle colonne (“nColumns”), la lista di record (“recordList”) e l’array di alberi di puntatori ai record che permettono di mantenere ordinati i record per ogni colonna. Infine contiene un puntatore heapReference !!!—GIACK—!!!.

È utile approfondire, nella struttura tabella, il funzionamento degli alberi di puntatori ai record e della lista di record. È oltretutto necessario chiarire che i record sono salvati fisicamente solo nella lista di record. Per ogni colonna della tabella è presente un red black tree, che tiene ordinato ogni record per ogni colonna della tabella. Nel momento dell’inserimento di un record, infatti, si inserisce un nodo RBT in ogni albero nella tabella, contenente il puntatore al nuovo record inserito in recordList.

Le rimanenti strutture dati sono alquanto facili da comprendere, ma se ne riporta una breve spiegazione in seguito.

```struct ParseResult``` ha un campo per ogni possibile informazione di cui può essere necessario fare il parse e un campo di controllo (```bool success```) che indica se esso è andato a buon fine o meno.

```struct QueryResultElement``` è una struttura che serve a contenere il risultato delle query. È una semplice linked list in cui c’è il campo “occurrence” utilizzato nel groupby per capire quante volte si ripete un elemento e il campo nodeValue punta al record a cui si vuol riferire questo nodo della lista.

**Corpo centrale**

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Leo integer malesuada nunc vel. Etiam erat velit scelerisque in dictum non consectetur. Tortor posuere ac ut consequat semper viverra nam libero justo. Dis parturient montes nascetur ridiculus mus mauris vitae. Arcu odio ut sem nulla pharetra diam. Vitae nunc sed velit dignissim sodales ut eu. Ornare aenean euismod elementum nisi quis. Mattis rhoncus urna neque viverra justo. Tortor at risus viverra adipiscing at in.

**Parser**

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Leo integer malesuada nunc vel. Etiam erat velit scelerisque in dictum non consectetur. Tortor posuere ac ut consequat semper viverra nam libero justo. Dis parturient montes nascetur ridiculus mus mauris vitae. Arcu odio ut sem nulla pharetra diam. Vitae nunc sed velit dignissim sodales ut eu. Ornare aenean euismod elementum nisi quis. Mattis rhoncus urna neque viverra justo. Tortor at risus viverra adipiscing at in.