Phase #2: Implementing RDF Database using Minibase

CSE 510: Database Management Systems Implementation

Arizona State University - Spring 2022

GROUP 10:
Sharanya Banerjee
Aishwarya Baalaji Rao
Manisha Nandkumar
Phadke
Lalit Kumar Tiwari
Manoj Mysore Srinath
Vivek Keshava

ABSTRACT: This document serves as the report to the second phase of the project. In this phase, we modify the underlying structure of Minibase which uses Tuples, to a new architecture using Quadruples. The Quadruples each consist of 4 fields, namely: Subject, Predicate, Object and Confidence for each record. We also incorporate a few functions into this architecture, namely: BatchInsert and Query to insert records into the database and retrieve records from the database respectively.

KEYWORDS: Minibase, Tuples, Quadruples, Entity, Label, Predicate, BTree, Disk Manager, Heap, Query, BatchInsert, Iterator, Page count, Read count, Write count

INTRODUCTION

The second phase of the project is intended to modify the underlying structure of Minibase which uses Tuples, to a new architecture using Quadruples. Each quadruple is formatted to have 4 fields: Subject, Object, Predicate and Confidence stored in some order selected by the user. The records are also indexed based on the user's choice. The system enables the user to insert data into the database using the *BatchInsert* command. Post insertion, the user can query the db using the *Query* command to retrieve records from the database. Depending on a few factors like duplicates in data and type of indexing, the number of Page counts, Read counts and Write counts will vary. We leverage several classes in Minibase like *BTree, Disk Manager, Global, Heap and Iterator* and have also used our own class *DBUtils* which contains classes to perform operations into the db. We also use the predefined *$makedb* class to compile the db prior to running commands.

DESCRIPTION OF IMPLEMENTATION

The assumptions made for implementing the RDF Database are:

1) If there exist quadruples having the same subject, object and predicate then only the quadruple with the highest confidence is stored.
2) For sorting records, the default order is subject, object, predicate, and confidence.
3) For the query program, "*" indicates a null filter.

The modification of tuple into quadruple involves adding a new field into the record, called confidence. The first stage is creating the quadruple ID (QID), label ID (LID), entity ID (EID) and predicate ID (PID) classes by altering the RID class which is used to identify each tuple in Minibase. Each of these classes consists of constructors to initialize an instance of the class and methods to copy an object, compare two objects of the class and a writeToByteArray method which writes the object into a byte array at offset. The QID class also contains the copyQID method, which makes a copy of the given QID. The LID class also contains methods to create a copy of the given LID, to return the corresponding EID of the object and to return the corresponding PID of the object it is invoked on. The EID and PID classes also contain the method returnLID which returns the corresponding LID.

In implementing the RDF database, records are stored as quadruples in quadruple heap files while entities and predicates are stored in label heap files. The quadrupleheap package is created by modifying the heap package to account for the extended quadruple construct. The major difference between tuple and quadruple is that a tuple is of variable length, while a quadruple has four fixed fields, namely, Subject (identified by

EID), Predicate (identified by PID), Object (identified by EID) and a Value (of type attrReal denoting confidence level). This difference is leveraged to modify the Tuple class to create the Quadruple class. The class THFPage is used to initialize a new page or open an existing THFPage object, and contains methods to return the data as a byte array, dump the contents of a page, return the PageId of the previous page or next page, set the PageId of the previous page or next page to a given value respectively. This class can also be used to insert or delete records with the specified QID, and to return the QID of the next record on the page. The class TScan initializes an object of type scan, which can perform a sequential scan to identify the next record, position the scan cursor to the location of the record with the specified QID, and move to the next data page in the file. The package labelheap is created in a similar manner, by modifying Tuple class to Label class.

In our project, the classes of the btree package are used to create data structures for indexing quadruples in QuadrupleHeapFiles and for indexing labels in LabelHeapFiles, identified by their QIDs and LIDs respectively. The BT class provides an interface to key and data abstraction, thereby providing a way to package and unpackage <key,value> pairs while maintaining efficiency in space utilization. It also contains a method to compare two keys, determine the length of a key, find the length of a <key,value> pair in a node of the btree, to convert a data entry into an array of bytes, and to print the structure of the B+ tree. Debugging can be done by displaying the structure of the B+ tree to identify the leaf nodes storing the key values and their corresponding data pointers to the disk file block.

The class diskmgr.DB is modified to diskmgr.rdfDB, which generates all the necessary files such as entity label heap file, predicate label heap file, quadruple heap file and index files based on the btree structure for data organization. Apart from the existing methods in the original diskmgr.DB class, the diskmgr.rdfDB class also consists of a constructor to initialize the RDF database by creating a QuadrupleHeapFile for storing quadruples, two LabelHeapFiles for storing entity labels and for storing subject labels. It also contains counters for returning the number of quadruples, entities, predicates, distinct subjects and distinct objects in the database. We used a colon (:) as a delimiter to isolate the fields in each record. There are also methods to ensure uniqueness of entity labels for entities and predicates by scanning the B+ tree repeatedly to check if the entity/predicate is present. If it is found then it returns the EID/PID of the label and if it does not exist in the btree, the key and lid of the label is inserted into the B+ tree and its EID/PID is returned. The openStream method is invoked by passing a subjectFilter, predicateFilter, objectFilter and confidenceFilter. The method returns a stream of quadruples, whose subject label is identical to subjectFilter, predicate label is identical to predicateFilter, object label is same as objectFilter, and confidence is greater than or equal to

the confidenceFilter. The order in which the stream of quadruples is returned depends on the orderType which is passed as an argument to the openStream method.
This is done by forming an unclustered BTree sorted heap file or index file on the labels as per the sorting scheme for the given orderType. This is performed after destroying the existing index by deleting the records having that key and QID.

Table 1. given below, correlates orderType to the sorting scheme used:

orderType specified	**Sorting Scheme Used**
1	subject label, predicate label, object label and then confidence
2	predicate label, subject label, object label and then confidence.
3	subject label followed by confidence
4	predicate label followed by confidence
5	object label followed by confidence
6	confidence

Table 1. Sorting Scheme

The diskmgr.Stream class provides various kinds of accesses to the quadruples in our RDF database. The constructor of this class comprises of an approach to handle null filters (represented by * while passing arguments) while initializing a stream of quadruples. For example, if the sorting scheme used is 3, the quadruples are sorted first in ascending order of subject labels. If there are identical subject labels, then the quadruples having the same subject label are then ordered by confidence.

The iterator.QuadrupleUtils class contains the method CompareQuadrupleWithQuadruple which returns 0 if the quadruples are equal in the field specified by the quadruple_fld_no, 1 if q1 is greater and -1 is q1 is smaller. If quadrupleOrder is passed as a parameter to the CompareQuadrupleWithQuadruple function, the quadruples are compared successively based on the sorting scheme specified in the above table. The class iterator.LabelUtils is formulated in a similar manner.

One of the prime goals of the projects is to count the number of reads and writes whenever the database is created or modified. This is done by the class pcounter.java where one counter is used for counting the number of reads (rcounter) and another counter is used for counting the number of writes (wcounter). The methods to increase the number of reads or writes are used in diskmgr.DB when a page is read into a byte array, or when a page is written into the disk.

batchinsert: We implemented the program batchinsert which is invoked by the command:

```
java dbUtils.BatchInsert DATAFILE INDEXOPTION RDFDBNAME
```

The program first checks if there is an existing DB and deletes all the records present in the DB in order to insert new records based on the index option given. The new data entered from the DATAFILE is first stored in a heap file (*tempresult*). The sorting occurs based on the index option chosen, specified in the INDEXOPTION variable. Based on this INDEXOPTION, we create a BTree file which will be used for retrieving the records later using indexing.

1. The QID values are inserted into the quadruple heap file.
2. The predicate values (labels) are inserted into the predicate heap file.
3. The label values, subjects and objects, are inserted into the entity heap file.

Index Options: The *tempresult* heap file is used to sort the quadruples according to the INDEXOPTION. The indexing options that we implemented are as follows:
Note: <key, value> ordered pair is used for all the indexing options given below. A quadruple BTree file is created (Quadruple_BTreeIndex) where these ordered pairs are stored for indexing.

1. **BTree index file on confidence**: The key in this case is confidence and the values are the matching QIDs formed as <confidence, QIDs>. Whenever there is a query request, the Quadruple_BTreeIndex file is used to fetch the matching QIDs which are indexed by the confidence key.
2. **BTree index file on subject and confidence**: The key in this case is a combination of subject and confidence, and the values are the matching QIDs formed as <subject:confidence, QIDs>. Whenever there is a query request, the Quadruple_BTreeIndex file is used to fetch the matching QIDs which are indexed by subject and confidence key.
3. **BTree index file on object and confidence**: The key for this particular case is a combination of object and confidence, and the values are the matching QIDs formed as <object:confidence, QIDs>. Whenever there is a query, the Quadruple_BTreeIndex file is used to fetch the matching QIDs which are indexed by object and confidence key.
4. **BTree index file on predicate and confidence**: The key for this particular case is a combination of predicate and confidence, and the values are the matching QIDs formed as <predicate:confidence,

QIDs>. Whenever there is a query based on predicate and confidence, the Quadruple_BTreeIndex file is used to fetch the matching QIDs which are indexed by predicate and confidence.

5. **BTree index file on predicate**: The key in this case is the predicate, and the values are the matching QIDs formed as <predicate, QIDs>. Whenever there is a query, the Quadruple_BTreeIndex file is used to fetch the matching QIDs which are indexed by the predicate key.

Table 2. given below, shows the Read/Write counts for various INDEXOPTIONS and SORT schemes:

INDEXOPTION	**SORTOPTION**	**Read Counter**	**Write Counter**
1	**1**	**1063396**	**183**
2	**3**	**176173**	**140**
3	**2**	**639118**	**179**
4	**2**	**570363**	**162**
5	**4**	**194204**	**142**

Table 2. Read/Write counts for all filters

Query: Our implementation of the program query involves accessing the database and printing out the relevant records in the order specified in INDEXOPTION in the command given below:

query RDFDBNAME INDEXOPTION ORDER SUBJECTFILTER PREDICATEFILTER OBJECTFILTER CONFIDENCEFILTER

The query is first parsed to distinguish each parameter and the records with the same subject label as SUBJECTFILTER, predicate label as PREDICATEFILTER, object label as OBJECTFILTER and confidence greater than or equal to CONFIDENCEFILTER from the database RDFDBNAME are returned. After running the query, the number of disk pages read and the number of disk pages written is returned.

We implemented a program which could be run via the command line to display statistics for the RDF database, such as the name of the RDF database, the size of the database, the size of the pages of the database, the number of pages, size of quadruple, total count of entities, subjects, predicates, objects, quadruples, names of the quadruple files, entity files, predicate files and their record count, and the page replacement policy used. The program also returns the number of disk pages read and the number of disk pages written on.

INTERFACE SPECIFICATION

Our interface is a command line format which is invoked by the following command:

java dbUtils.BatchInsert tests/8-\ phase2_test_data.txt 1 "db14"

For Query also, we use the command line format to run the query. It is invoked by the following command:

java dbUtils.Query "db14" 1 1 "" "" "" "" 50

INSTALLATION AND RUNTIME

To install and run the system, we extract the zip file. Then, the following commands are run in terminal:

1. *cd ./src* - in Terminal
2. *$make db* - Compile all the java classes and create .class files for all packages
3. *java dbUtils.BatchInsert DATAFILEFILE INDEXOPTION RDFDBNAME* - Execute this command to batch insert all the data from any text file consisting of inputs, followed by the indexing option, followed by the db name. Eg: *java dbUtils.BatchInsert tests/8-\ phase2_test_data.txt 1 "vkrdfdb1_5"*
4. *query RDFDBNAME INDEXOPTION ORDER SUBJECTFILTER PREDICATEFILTER OBJECTFILTER CONFIDENCEFILTER NUMBUF* - Execute this command to query the database. The command contains the query keyword followed by the db name, indexoption, sorting order, filter for subjects, predicates, objects and confidence. We also specify the maximum number of records we want to retrieve as numbuf. Eg: *"vkrdfdb1_5" 5 5 "X" "name" "Y" "*" 50*
5. *report RDFDBNAME INDEXOPTION* - Run this command to provide the statistics of the database like Subject count, Object count, Number of pages, etc.

RELATED WORK

To understand the Minibase architecture, we spent some time going through the Minibase documentation from their homepage, in addition to our work in Phase 1. We also spend some time understanding what indexing using B and B+ trees is and how we can leverage different indexing schemes in our system. This also entailed a comprehensive analysis of the differences between BTrees and B+Trees and how they are populated in a Bottom-up architecture. The links to these materials are listed in the bibliography section.

CONCLUSION

From our above system description, we can get a fair idea about the systems architecture. As we can see from the directives, we are successfully able to insert records into the db as well as query information from the db based on user requirements. We can also retrieve information per run like page, read and write counts and number of subjects, objects, predicates, etc. and perform high-level analysis on them to identify root causes for the varying count values. This concludes the second phase of the project.

BIBLIOGRAPHY

1. Source file: ../../users/sharanyabanerjee/Desktop/Assignments/CSE510 DBMSI/minjava/javaminibase/src/javadoc/overview-frame.html (Hosted on local system).
2. https://research.cs.wisc.edu/coral/minibase/minibase.html
3. https://research.cs.wisc.edu/coral/minibase/project.html
4. https://stackoverflow.com/questions/1108/how-does-database-indexing-work
5. https://dzone.com/articles/database-btree-indexing-in-sqlite
6. https://www.tutorialspoint.com/unix_commands/make.html

APPENDIX

Individual Contribution:

1. **Sharanya Banerjee -** QID, LID, Modifying BTree package, Query
2. **Aishwarya Baalaji Rao -** PID, EID, rdfDB, Query
3. **Manisha Nandkumar Phadke -** THFPage, TScan, iterator.QuadrupleUtils, BatchInsert
4. **Lalit Kumar Tiwari -** QuadrupleHeapFile, Quadruple, diskmgr.Stream, Query
5. **Manoj Mysore Srinath -** LHFPage, LScan, iterator.LabelUtils, BatchInsert
6. **Vivek Keshava -** LabelHeapFile, Label, PCounter, BatchInsert
7. **Report and Presentation -** All team members