

Lab2 Write Up:

Yuanfeng Li

Feb.3rd.2021

1. Report Runtimes of the queries from Exercise 7

I had a lot of fun making the hash join function in this practice and I do have a lot to talk about, but making it short, I will give you a quick view of comparison.

Loops/Runtimes	Query 1	Query 2	Query 3
Nested Loop Join	0.3 seconds	0.69 seconds	> 500 seconds
Hash Join	0.19 seconds	0.64 seconds	1.18 seconds

(I putted the screen shots in the appendix's runtime section.)

- There are 4 things to notice:
 1. The hash join only works on equal operations. (Predicate.Op.EQUALS)
 2. The hash Join is more than 500 times faster than the nested loop on equal operations!! Woohoo!!
 3. I could not take the screenshots for nested loop join of query 3 since I accidentally turn off my laptop, so you will not find it in appendix section, but I do remember the runtime is around 530 seconds.
 4. All the runtimes are from the results in my laptop's Linux environment, not on attu3, since I am not a CSE major's student, so I am not 100% sure whether the speed will increase or decrease on attu3.

2. Describe what this lab was about?

In Summary I had implemented two major operation function including filter and join, and also the function to modify table such as insert Tuples, and delete Tuples.

I will give you a short intro on each cluster of classes that I developed.

1) Filter and Join

Filter only returns the tuples that meet the requirement of the predicate by using the compare method in , and join function joins the compare method in field interface, and join method using join both tuples from child 1 and child 2, with the requirement and condition of the Join Predicate.

2) Aggregates

Three java classes basically did the same thing, they aggregate the tuples, or even group by the tuples and fields with MIN, MAX, SUM, AVG, and COUNT operands.

3) HeapFile Mutability and Insertion and deletion

HeapFile calls buffer pool to manipulate the tables with insertion and deletion, and heappage do the lowest level job. If we insert and delete a tuple or some tuple, the heapPage and HeapFile would marked as dirty and stored in the bufferpool, then write to disk later for updates.

4) Page eviction

When we reach the limit of pages that can store in the buffer pool, we need to call flushpage, or flushAllpages in buffer pool to get more memory space in buffer pool. And page eviction is the tool to achieve that goal.

3. Design decisions

I will talk about my design in the Join.java first and then the page eviction policy:

For my Join.java:

My design was enlightened by the 3 queries in the optional contest. I was thinking using the nested loop join for all of them at first, but find it was really difficult to finish the 3rd query in short time. To be honest, I was eager to only pass the 3rd queries in short time, so the easiest method I can think of is to use a hash join method only for Equal operations. So you can see the hashjoin method in my Join.java open method, that I used it only when operation is EQUALS. Other than EQUALS, I will still use nested loops since hash function can not solve the greater joins or less joins, etc easily. Due to the short time of hand, I have not developed a merge sort join method which might be able to solve all kinds operations, including greater than or less than, etc.;

My Hash join method was focusing on putting the tuples from child1 into a hashmap, and use their hashcodes of the field value as the keys, and an array of tuple as values. After we put all the child1's tuples in the hashmap, we can iterate through the tuples from child2 and find out the arrays by using the hashcode of child2's field in the map, we can easily find out the arrays of child1 tuples that have the same hashcode with child2's, and then we can join child2's tuple with each child1's tuple in the arraylist and add the combined tuple in a global arraylist of tuple for future uses. I will just introduce the high level of implementation here, but you can find more specific details in my Join.java.

For Eviction Policy:

My design for the evictPage method is quite simple. I will always throw the first page in the keyset() of the cache, since I am using a HashMap for cache, so It does not really matter which page I flush. As long As I can get more memory in cache, it will be good enough.

4. One example of unit test

I found it was really difficult to understand how my joins work with exceptions using the current unit tests, so the one example of unit test I would suggest to add would be:

A unit test to test errors that might happen such as TransactionAbortedException of DBExceptions while we are joining tables.

5. Discuss and justify any changes you made to the API.

No changes.

6. Describe any missing or incomplete elements of your code.

As I mentioned in design decisions, I would like to develop an sort and merge join method for join.java, but due to the short time in hand, I couldn't really finish it before due date.

Appendix:

Nested Loop Join:

- Nested loop query 1

The terminal window shows the following commands and output:

```
lyf99@ubuntu:~/JW_lyf99/CSE_classes/cse444/simple-db-lyf99$ ls
authors.dat      file1.dat      some_data_file.dat
bin              file1.txt      some_data_file.txt
build.xml        lib            src
catalog.txt      log            test
'CSE444-lab1_write_up_Yuanfeng Li.pdf'  out            testWin10_emptyfileWin10
data.dat         paperauths.dat  turnInLab.sh
data.txt         papers.dat      venues.dat
dblp_simpledb.schema  README.md      simple-db-lyf99.tml
dist             simple-db-lyf99.tml

lyf99@ubuntu:~/JW_lyf99/CSE_classes/cse444/simple-db-lyf99$ ant dist
Buildfile: /home/lyf99/JW_lyf99/CSE_classes/cse444/simple-db-lyf99/build.xml

compile:
[depend] Deleted 11 out of date files in 0 seconds.
[javac] Compiling 11 source files to /home/lyf99/JW_lyf99/CSE_classes/cse444/simple-db-lyf99/bin/
src
[javac] warning: [options] bootstrap class path not set in conjunction with -source 8
[javac] Note: Some input files use or override a deprecated API.
[javac] Note: Recompile with -Xlint:deprecation for details.
[javac] 1 warning
dist:
[jar] Building jar: /home/lyf99/JW_lyf99/CSE_classes/cse444/simple-db-lyf99/dist/simpledb.jar

BUILD SUCCESSFUL
Total time: 1 second
lyf99@ubuntu:~/JW_lyf99/CSE_classes/cse444/simple-db-lyf99$ java -jar dist/simpledb.jar parser dblp_schema
SimpleDB> FROM papers p
SimpleDB> WHERE p.title LIKE 'selectivity';
Started a new transaction tid = 0
Added scan of table p
Added select list field p.title
p.title
-----
Optimizing ethanol production selectivity.
Development of feedforward receptive field structure of a single cell and its contribution to the orientation selectivity: a mod
Influences of formant bandwidth and auditory frequency selectivity on identification of place of articulation in stop consonants
A theoretical entropy score as a single value to express inhibitor selectivity.
ASH structure alignment package: Sensitivity and selectivity in domain classification.
5 rows.
Transaction 0 committed.
-----
0.30 seconds
SimpleDB>
```

The web browser shows the SimpleDB website with the following text:

We have built a SimpleDB-encoded version of the DBLP database; the needed files are located at http://www.cs.washington.edu/education/courses/cse444/17wi/labs/lab2/dblp_data.tar.gz.

You should download the file and unpack it. It will create four files in the `dblp_data` directory. Move them into your repository directory. The following commands will accomplish this, if you execute them at the root of your repository:

```
$ wget http://www.cs.washington.edu/education/courses/cse444/17wi/labs/lab2/dblp_data.tar.gz
$ tar xzf dblp_data.tar.gz
$ mv dblp_data/* .
$ rm -r dblp_data.tar.gz dblp_data
```

NOTE:

- Make sure to first compile (i.e., run `ant dist`).
- Make sure the filepath of the project in which you are working contains no special characters (like spaces), as it will cause an error while running the parser.

You can then run the parser with the following command:

```
$ java -jar dist/simpledb.jar parser dblp_simpledb.schema
```

Execute the three queries below using your SimpleDB prototype and report the times in your lab write-up.

Query 1

```
SELECT p.title
FROM papers p
WHERE p.title LIKE 'selectivity';
```

Query 2

```
SELECT p.title, v.name
FROM papers p, authors a, paperauths pa, venues v
```

- Nested loop query 2

The terminal window shows the following commands and output:

```
SimpleDB> SELECT p.title, v.name
SimpleDB> FROM papers p, authors a, paperauths pa, venues v
SimpleDB> WHERE a.name = 'E. F. Codd'
SimpleDB> AND pa.authorid = a.id
SimpleDB> AND pa.paperid = p.id
SimpleDB> AND p.venueid = v.id;
Started a new transaction tid = 1
Added scan of table p
Added scan of table a
Added scan of table pa
Added scan of table v
Added join between pa.authorid and a.id
Added join between pa.paperid and p.id
Added join between p.venueid and v.id
Added select list field p.title
Added select list field v.name
p.title v.name
-----
Further Normalization of the Data Base Relational Model., IBM Research Report San Jose California
Interactive Support for Non-Programmers: The Relational and Network Approaches., IBM Research Report San Jose California
Data Base Sublanguage Founded on the Relational Calculus., IBM Research Report San Jose California
Relational Completeness of Data Base Sublanguages., In: R. Rustin (ed.): Database Systems: 65-98 Pren
tice Hall and IBM Research Report RJ 987 San Jose California
Derivability Redundancy and Consistency of Relations Stored in Large Data Banks., IBM Research Report San Jose California
The Capabilities of Relational Database Management Systems., IBM Research Report San Jose California
Seven Steps to Rendezvous with the Casual User., IFIP Working Conference Data Base Management
Normalized Data Base Structure: A Brief Tutorial., IBM Research Report San Jose California
The Gamma-0 n-ary Relational Data Base Interface Specifications of Objects and Operations., IBM Research Report
RENDZVOUS Version 1: An Experimental English Language Query Formulation System for Casual Users of R
elational Data Bases., IBM Research Report
Universal Relation Fails to Replace Relational Model (letter to the editor), IEEE Software
11 rows.
Transaction 1 committed.
-----
0.69 seconds
SimpleDB>
```

The web browser shows the SimpleDB website with the following text:

You can then run the parser with the following command:

```
$ java -jar dist/simpledb.jar parser dblp_simpledb.schema
```

Execute the three queries below using your SimpleDB prototype and report the times in your lab write-up.

Query 1

```
SELECT p.title
FROM papers p
WHERE p.title LIKE 'selectivity';
```

Query 2

```
SELECT p.title, v.name
FROM papers p, authors a, paperauths pa, venues v
WHERE a.name = 'E. F. Codd'
AND pa.authorid = a.id
AND pa.paperid = p.id
AND p.venueid = v.id;
```

Query 3

```
SELECT a2.name, count(p.id)
FROM papers p, authors a1, authors a2, paperauths pa1, paperauths pa2
WHERE a1.name = 'Michael Stonebraker'
AND pa1.authorid = a1.id
AND pa1.paperid = p.id
AND pa2.authorid = a2.id
AND pa1.paperid = pa2.paperid
```

- Nested loop query3: More than 500 seconds!(as mentioned earlier, I forgot to take the screenshots, but I remember it was around 500 seconds)

Hash Join:

Query 1: using hash Join for Equal, and nested loop Join for other than equal. 0.19s

```
lyf99@ubuntu: ~/UW_lyf99/CSE_classes/cse444/simple-db-lyf99
SimpleDB> WHERE p.title LIKE 'selectivity';
Started a new transaction tid = 7
Added scan of table p
Added select list field p.title
p.title
-----
Optimizing ethanol production selectivity.
Development of feedforward receptive field structure of a single cell and its contribution to the ori
entation selectivity: a mod
Influences of formant bandwidth and auditory frequency selectivity on identification of place of art
iculation in stop consonants
A theoretical entropy score as a single value to express inhibitor selectivity.
ASH structure alignment package: Sensitivity and selectivity in domain classification.
5 rows.
Transaction 7 committed.
-----
0.19 seconds
SimpleDB>
```

Query 2: 0.64 s

```
lyf99@ubuntu: ~/UW_lyf99/CSE_classes/cse444/simple-db-lyf99
SimpleDB> SELECT p.title, v.name
SimpleDB> FROM papers p, authors a, paperauths pa, venues v
SimpleDB> WHERE a.name = 'E. F. Codd'
SimpleDB> AND pa.authorid = a.id
SimpleDB> AND pa.paperid = p.id
SimpleDB> AND v.venueid = v.id;
Started a new transaction tid = 11
Added scan of table p
Added scan of table a
Added scan of table pa
Added scan of table v
Added join between pa.authorid and a.id
Added join between pa.paperid and p.id
Added join between v.venueid and v.id
Added select list field p.title
Added select list field v.name
p.title v.name
-----
The Gamma-0 n-ary Relational Data Base Interface Specifications of Objects and Operations., IBM Resear
ch Report
Interactive Support for Non-Programmers: The Relational and Network Approaches., IBM Research Report
San Jose California
Universal Relation Falls to Replace Relational Model (letter to the editor), IEEE Software
Derivability, Redundancy and Consistency of Relations Stored in Large Data Banks., IBM Research Report
San Jose California
Relational Completeness of Data Base Sublanguages., In: R. Rustin (ed.): Database Systems: 65-98 Prentice
Hall and IBM Research Report R3 987 San Jose California
RENDZOUS Version 1: An Experimental English Language Query Formulation System for Casual Users of R
elational Data Bases., IBM Research Report
Further Normalization of the Data Base Relational Model, IBM Research Report San Jose California
Normalized Data Base Structures: A Brief Tutorial., IBM Research Report San Jose California
Data Base Sublanguage Founded on the Relational Calculus., IBM Research Report San Jose California
Seven Steps to Rendezvous with the Casual User., IFIP Working Conference Data Base Management
The Capabilities of Relational Database Management Systems., IBM Research Report San Jose California
11 rows.
Transaction 11 committed.
-----
0.64 seconds
SimpleDB>
```

Query 3: 1.18 seconds

```
lyf99@ubuntu: ~/UW_lyf99/CSE_classes/cse444/simple-db-lyf99
SimpleDB> SELECT a2.name, count(p.id)
SimpleDB> FROM papers p, authors a1, authors a2, paperauths pa1, paperauths pa2
SimpleDB> WHERE a1.name = 'Michael Stonebraker'
SimpleDB> AND pa1.authorid = a1.id
SimpleDB> AND pa1.paperid = p.id
SimpleDB> AND pa2.authorid = a2.id
SimpleDB> AND pa1.paperid = pa2.paperid
SimpleDB> GROUP BY a2.name
SimpleDB> ORDER BY a2.name;
Started a new transaction tid = 10
Added scan of table p
Added scan of table a1
Added scan of table a2
Added scan of table pa1
Added scan of table pa2
Added join between pa1.authorid and a1.id
Added join between pa1.paperid and p.id
Added join between pa2.authorid and a2.id
Added join between pa1.paperid and pa2.paperid
GROUP BY FIELD : a2.name
Aggregate field is p.id, agg fun is : count
Added select list field a2.name
a2.name p.id
-----
Akhil Kumar,1
Dale Skeen,1
Eric N. Hanson,1
Lawrence A. Rowe,1
Michael Hirohama,1
Michael Stonebraker,8
Spyros Potamianos,1
7 rows.
Transaction 10 committed.
-----
1.18 seconds
```