Database System Implementation CSCI 421 Group Project Phase 4

v1.0

1 Phase Description

This is the forth, and final, phase of the semester long group project; other than the individual group evaluations.

In this phase you will be implementing indexing.

There are a few basic rules when implementing this phase:

- You must create and use an instance of the storage manager you created in phase 1.
- You must create and use an instance of the DDL/DML parsers you created in prior phases.
- You can assume the database location path exists and is accessible.
- You must use the catalog structure implemented in prior phases.
- When updating/removing/altering/inserting an attribute that has an index, you must update the index to reflect that change.
- Depending on your implementation you most likely will need to make changes to much of the functionality for the prior phases.

2 Phase Layout

In this you will be implementing B+ tree based indexing.

How you implement this into your project is up to you.

The program will take an additional command line argument: `<indexing`.

If set to `true`, the system will create and use indixies. If set to anything else it will not use indices.

- Python: `python main.py <db_loc> <page_size> <buffer_size> <indexing>`
- Java: `java Main <db_loc> <page_size> <buffer_size> <indexing>`
- C: `./main <db_loc> <page_size> <buffer_size> <indexing>`

3 B+ Tree Index

You will implement a B+ Tree based index; based on lecture and book algorithms.

- Each node in the tree will be a page on hardware stored in a index file. Very similar to how tables are implemented.
- Each node will have to track if it is an internal node or a leaf node.
- Internal nodes should contain a pointer to other nodes (pages)
- Leaf nodes should contain pointers to buckets (special pages).
- A root node can be either an internal node or a leaf node.

- A search-key value can only be one attribute value.
- The number of pointers in the node, the N of the tree, will be the max page size divided by the max size of a search-key, page pointer pair size. This will then be floored and minus one from the floor. The size of the search-key is the maximum size of any value the search-key can be.
  Example:

```
Page size is 4096 bytes
Data Type is varchar(10) --> max size is 10 * 2 bytes = 20 bytes
Page pointer is an integer --> size 4 bytes
Pair size = 20 + 4 = 24 bytes
4096 / 24 = 170.6667 pairs
floor(170.6667) - 1 = 169 --> N value of the B+ Tree
```

- Buckets are special pages that contain `<page_number, index>` pairs. The page number references a page containing an entry with that search key value. The index references the index on that page containing an entry with that search key value. The size of bucket page will be the `floor( page_size / ((size(integer) + size(integer))`. The last entry in the page will be reserved as a special pointer to the next page of the bucket; it will be empty on the last page. Buckets in B+ Trees cannot contain pairs with different search key values.

4 Using an Index

When accessing data in the database you must you index if one exists on the search-key attribute combination. You must also update the index as you insert/update/delete data in the database.

4.1 Inserting Data using a Primary Index

When inserting/deleting/updating data in the database you must use the primary key index to find the page of the entry.

When inserting you will find the location in the index that the new item belongs and look at the page number of the entry just before it. Then try to insert it into that page.

For example, if inserting an row with the primary key of 5. Navigate to the node in the tree that the entry would belong in. If there is a search key with that value this is an errors (this is the primary key index). If there is no search key of that value insert the search key value; lets say between search keys 4 and 8. Get the `<page_number, index>` of pair of the search key 4 just before this new entry. This will give you an idea of where to insert the new entry, because 5 will belong right after 4.

Insert the new record data into the table page right after the `<page_number, index>` of the prior search key value. You must follow the rules of the storage manger; such as splitting, ordering, etc.

Then insert the new search key and record pointer into the B+Tree; updating the tree as needed.

You may need to update other search key's pointers if the insertion of the new data caused the storage manager to move rows to a new page when splitting pages.

This will require changes to your Storage Manager.

4.2 Deleting Data using a Primary Index

When deleting a value from the B+Tree, you will find the search key in thew tree; if it does not exist there is nothing to delete.

If the search key exists there should only be one record pointer at that search key value. Get the record pointer data and delete that record from that `<page_number, index>` location. Finally, delete the search key from the tree.

This will require changes to your Storage Manager.

4.3 Updating Data using a Primary Index

Updating a search key will involve inserting a new search key an deleting a the old one if the primary key or location changed.

This will require changes to your Storage Manager.

5 Testing

The indexing will be tested in two phases:

- with indexing off
- with indexing on

In each chase there will be a large amount of data (thousands of rows) inserted/deleted/updated. There should be a noticeable performance difference when indexing is turn on.

6 Project Constraints

This section outlines details about any project constraints or limitations.

Constraints/Limitations:

- You must follow the rules of B+ Trees as outlined in lecture.
- Everything, except for the values in Strings (char and varchar), is case-insensitive; like SQL. Anything in double quotes is to be considered a String. String literals will be quoted and can contain spaces. Example: `"foo bar"` is a string literal. `foo` is a name; not a string literal. The quotes do not get stored in the database. They must be added when printing the data.

- Your project must run and compile on the CS Linux machines.
- Submit only your source files in the required file structure. Do not submit and IDE directories or projects.
- Your code must run with any provided tests cases/code. Failure to do so will result heavy penalties.
- Words such as `Integer`, `create`, `table`, `drop`, etc are considered key words and cannot be used in any attribute or table name.
- Data must be checked for validity. Examples:
  - type matching
  - not nulls
  - primary keys
- Any changes to data in the database must keep the database consistent. This means that all not null, foreign keys, and primary keys must be observed.

7 Grading

Your implementation will be grading according to the following:

- (60%) B+Tree based indices.
- (40%) Indices added for primary keys.

Penalties:

- (-50% of points earned) Data is written as text not binary data.
- (-25% of points earned) Does not use storage manager to handle hardware access. Catalog reading/writing to hardware does not need to be handled by the storage manager.
- (up to -50% of points earned) Does not work with any provided testing files.
- (up to -100% of points earned) Does not compile on CS machines.

Penalties will be based on severity and fix-ability. The longer it takes the grader to fix the issues the higher the penalty.

Examples:

- Code does not compile due to missing semicolon: -5%
- Code does not compile/run with testers due to missing functions or un-stubbed functions: -10%
- Multiple syntax errors causing issues compiling issues and takes over 30 mins to fix: -100%
- Multiple Crashes with provided testers that take an extended time to fix: -50%

These are just examples.The basic idea is "Longer to fix, more points lost. Eventually give up, assign a zero."

8 Submission

Zip any code that your group wrote in a file called `phase4.zip`. Maintain any required package structure. Do NOT submit any provided code. The grader will use their own versions.

Submit the zip file to the Phase 1 Assignment box on myCourses. No emailed submissions will be accepted. If it does not make it in the proper box it will not be graded.

The last submission will be graded.

There will be a 72 hour late window. During this late window no questions will be answered by the instructor. No submissions will be accepted after this late window.