PROJECT #2 - EXTENDIBLE HASH INDEX

Do not post your project on a public Github repository.

OVERVIEW

The second programming project is to implement a disk-backed hash table for the BusTub DBMS. Your hash table is responsible for fast data retrieval without having to search through every record in a database table.

You will need to implement a hash table using the extendible hashing hashing scheme.

This index comprises a directory page that contains pointers to bucket pages. The table will access pages through your bu pool from Project #1. The table contains a directory page that stores all the metadata for the table and buckets. Your hash table needs to support bucket splitting/merging for full/empty buckets, and directory expansion/contraction for when the global depth must change.

You will need to complete the following tasks in your hash table implementation:

- Page Layouts
- Extendible Hashing Implementation
- Concurrency Control

This is a single-person project that will be completed individually (i.e., no groups).

Release Date: Sep 27, 2021
Due Date: Oct 21, 2021 @ 11:59pm

PROJECT SPECIFICATION

Like the first project, we are providing you with stub classes that contain the API that you need to implement. You should modify the signatures for the pre-defined functions in these classes. If you do this, then it will break the test code that we use to grade your assignment you end up getting no credit for the project. If a class already contains certain member varia you should **not** remove them. But you may add private helper functions/member variables to these classes in order to cor realize the functionality.

The correctness of the extendible hash table index depends on the correctness of your buffer pool implementation. We wi provide solutions for the previous programming projects.

TASK #1 - PAGE LAYOUTS

you create a hash table, write its pages to disk, and then restart the DBMS, you should be able to load back the hash table disk after restarting.

To support reading/writing hash table buckets on top of pages, you will implement two `Page` classes to store the data of y hash table. This is meant to teach you how to allocate memory from the `BufferPoolManager` as pages.

- Hash Table Directory Page
- Hash Table Bucket Page

HASH TABLE DIRECTORY PAGE

This class holds all of the meta-data for the hash table. It is divided into the fields as shown by the table below:

Variable Name	Size	Description
`page_id_`	4 bytes	Self Page Id
`lsn_`	4 bytes	Log sequence number (Used in Project 4)
`global_depth_`	4 bytes	Global depth of the directory
`local_depths_`	512 bytes	Array of local depths for each bucket (uint8)
`bucket_page_ids_`	2048 bytes	Array of bucket `page_id_t`

The `bucket_page_ids_` array maps bucket ids to `page_id_t` ids. The ith element in `bucket_page_ids_` is the `page_id` for ith bucket.

You must implement your Hash Table Directory Page in the designated files. You are only allowed to modify the directory (`src/include/storage/page/hash_table_directory_page.h`) and its corresponding source file (`src/storage/page/hash_table_directory_page.cpp`). The Directory and Bucket pages are fully separate from the LinearProbeHashTable's Header and Block pages, so please make sure that you are editing the correct files.

HASH TABLE BUCKET PAGE

The Hash Table Bucket Page holds three arrays:

- \`occupied_\` : The ith bit of \`occupied_\` is 1 if the ith index of \`array_\` has ever been occupied.
- \`readable_\` : The ith bit of \`readable_\` is 1 if the ith index of \`array_\` holds a readable value.
- \`array_\` : The array that holds the key-value pairs.

The number of slots available in a Hash Table Bucket Page depends on the types of the keys and values being stored. You c need to support fixed-length keys and values. The size of keys/values will be the same within a single hash table instance, b you cannot assume that they will be the same for all instances (e.g., hash table #1 can have 32-bit keys and hash table #2 c have 64-bit keys).

You must implement your Hash Table Bucket Page in the designated files. You are only allowed to modify the header file (`src/include/storage/page/hash_table_bucket_page.h`) and its corresponding source file (`src/storage/page/hash_table_bucket_page.cpp`). The Directory and Bucket pages are fully separate from the LinearProbeHashTable's Header and Block pages, so please make sure that you are editing the correct files.

Each Hash Table Directory/Bucket page corresponds to the content (i.e., the byte array `data_`) of a memory page fetched buffer pool. Every time you try to read or write a page, you need to first fetch the page from buffer pool using its unique `page_id`, then reinterpret cast to either a directory or a bucket page, and unpin the page after any writing or reading operations.

Bucket Page: - Insert - Remove - IsOccupied - IsReadable - KeyAt - ValueAt

Directory Page: - GetGlobalDepth - IncrGlobalDepth - SetLocalDepth - SetBucketPageId - GetBucketPageId

You are free to design and implement additional new functions as you see fit. However, you must be careful to watch for n collisions. These should be rare, but would arise in Gradescope as a compiler error.

TASK #2 - HASH TABLE IMPLEMENTATION

You will implement a hash table that uses the extendible hashing scheme. It needs to support insertions (`Insert`), point se (`GetValue`), and deletions (`Remove`). There are many helper functions either implemented or documented the extendible h table's header and cpp files. Your only strict API requirement is adhering to `Insert` , `GetValue` , and `Remove` . You also mus leave the VerifyIntegrity function as it is. Please feel free to design and implement additional functions as you see fit.

Your hash table must support both unique and non-unique keys. Duplicate values for the same key are not allowed. This m that `(key_0, value_0)` and `(key_0, value_1)` can exist in the same hash table, but not `(key_0, value_0)` and `(key_0, value_0)` . The `Insert` method only returns false if it tries to insert an existing key-value pair.

Your hash table implementation must hide the details of the key/value type and associated comparator, like this:

```
template <typename KeyType, typename ValueType, typename KeyComparator>
class ExtendibleHashTable {
   // ---
};
```

These classes are already implemented for you:

- `KeyType`: The type of each key in the hash table. This will only be GenericKey, the actual size of GenericKey is specifie and instantiated with a template argument and depends on the data type of indexed attribute.
- `ValueType`: The type of each value in the hash table. This will only be 64-bit RID.
- `KeyComparator`: The class used to compare whether two KeyType instances are less/greater-than each other. These be included in the KeyType implementation files. Variables with the `KeyComparator` type are essentially functions; fo instance, given two keys `KeyType key1` and `KeyType key2`, and a key comparator `KeyComparator cmp`, you can com the keys via `cmp(key1, key2)`.

Note that you can equality-test ValueType instances simply using the `==` operator.

EXTENDIBLE HASHING IMPLEMENTATION DETAILS

This implementation requires that you implement bucket splitting/merging and directory growing/shrinking. Some implementations of extendible hashing skip the merging of buckets as it can cause thrashing in certain scenarios. We implement it here to provide a full understanding of the data structure and provide more opportunities for learning how t manage latches, locks, page operations (fetch/pin/delete/etc).

DIRECTORY INDEXING

When inserting into your hash index, you will want to use the **least-significant bits** for indexing into the directory. Of cour is possible to use the most-significant bits correctly, but using the least-significant bits makes the directory expansion operation much simpler.

SPLITTING BUCKETS

provided API is more amenable to this approach. As always, you are welcome to factor your own internal API.

MERGING BUCKETS

Merging must be attempted when a bucket becomes empty. There are ways to merge more aggressively by checking the occupancy of buckets and their split images, but these expensive checks and extra merges can increase thrashing.

To keep things relatively simple, we provide the following rules for merging:

Only empty buckets can be merged.
Buckets can only be merged with their split image if their split image has the same local depth.
Buckets can only be merged if their local depth is greater than 0.

If you are confused about a "split image," please review the algorithm and code documentation. The concept falls out quite naturally.

DIRECTORY GROWING

There are no fancy rules for this. You either have to grow the directory, or you don't.

DIRECTORY SHRINKING

Only shrink the directory if the local depth of every bucket is strictly less than the global depth of the directory. You may s other tests for the shrinking of the directory, but this one is trivial since we keep the local depths in the directory page.

PERFORMANCE

An important performance detail is to only take write locks and latches when they are needed. Always taking write locks w inevitably timeout on Gradescope.

In addition, one potential optimization is to factor your own scans types over the bucket pages, which can avoid repeated s in certain cases. You will find that checking many things about the bucket page often involves a full scan, so you can potent collect all this information in one pass.

TASK #3 - CONCURRENCY CONTROL

Up to this this point you could assume that your hash table only supported single-threaded execution. In this last task, you modify your implementation so that it supports multiple threads reading/writing the table at the same time.

You will need to have latches on each bucket so that when one thread is writing to a bucket other threads are not reading modifying that index as well. You should also allow multiple readers to be reading the same bucket at the same time.

You will need to latch the whole hash table when you need to split or merge buckets, and when global depth changes.

REQUIREMENTS AND HINTS

LATCHES

There are two latches to be aware of in this project. The first is `table_latch_` in `extendible_hash_table.h`, which takes latches on the extendible hash table. This comes from the RWLatch class in `src/include/common/rwlatch.h`. As you can s the code, it is backed by `std::mutex`. The second is the built-in page latching functionality in `src/include/storage/page.` This is what you must use to protect your bucket pages. Note that to take a read-lock on the `table_latch_` you call `RLock` from `RWLatch.h`, but to take a read-lock on a bucket page you must `reinterpret_cast<Page *>` to a page pointer, and cal `RLatch` method from `page.h`.

Project 4 will explore concurrency control through locking with the LockManager in `LockManager.h`. You do **not** need the LockManager at all for this project.

TRANSACTION POINTER

You can simply pass `nullptr` as the Transaction pointer argument when it is required. This `Transaction` object comes frc `src/include/concurrency/transaction.h`. It provides methods to store the page on which you have acquired latch while traversing through the Hash Table. You do not need to do this to pass the tests.

INSTRUCTIONS AND FAQ

DEV ENVIRONMENT - SETUP

See the Project #0 instructions on how to create your private repository and setup your development environment.

⚠ You must pull the latest changes on our BusTub repo for test files and other supplementary files v have provided for you. Run `git pull public master`.

COMPILATION ISSUES

If you have trouble compiling, try deleting your build directory and making a fresh one. Running `make clean` does not completely reset the compilation process, so starting from scratch with `cmake ..` can help.

TESTING - GTEST AND GDB

You can test the individual components of this assigment using our testing framework. We use GTest for unit test cases.

```
cd build
make -j hash_table_test
./test/hash_table_test
```

We encourage you to use `gdb` to debug your project if you are having problems. Here is a nice reference for useful comma in `gdb`. Make sure you've ran cmake with debug flags on so `gdb` has debugging symbols to use.

MEMORY SAFETY - VALGRIND

To ensure your implementation does not have memory leak, you can run the test with Valgrind. Make sure that you run Valgrind **without** the cmake debug flags, as this sometime causes ASan runtime errors. If you have debug mode on, you car either remake your build directory, or have two separate build directories.

```
cd build
make -j hash_table_test
valgrind --trace-children=yes \
   --leak-check=full \
   --track-origins=yes \
   --soname-synonyms=somalloc=*jemalloc* \
   --error-exitcode=1 \
   --suppressions=../build_support/valgrind.supp \
   ./test/hash_table_test
```

Important: These tests are only a subset of the all the tests that we will use to evaluate and grade your project. You shoulc write additional test cases on your own to check the complete functionality of your implementation. In addition, if Valgrinc times out then it's possible that your implementation is not efficient enough. If your buffer pool manager implementation v slow, then you might need to debug the BPM as well.

Your code must follow the Google C++ Style Guide. We use Clang to automatically check the quality of your source code. project grade will be zero if your submission fails any of these checks.

Execute the following commands to check your syntax. The `format` target will automatically correct your code. The `check-lint` and `check-clang-tidy` targets will print errors and instruct you how to fix it to conform to our style guide.

```
$ make -j format
$ make -j check-lint
$ make -j check-clang-tidy
```

If you have issues with any of these commands, it may relate to either your python installation, or file permissions on your remote files. Please try to read the error, understand the problem, and resolve it by doing things like inspecting and editin your `$PATH`, managing permissions with `chmod`, and managing installed packages with `brew` or `apt`.

DEVELOPMENT HINTS

Instead of using `printf` or `cout` statements for debugging, use the `LOG_*` macros for logging information like this:

```
LOG_INFO("# Pages: %d", num_pages);
LOG_DEBUG("Fetching page %d", page_id);
```

To enable logging in your project, you will need to reconfigure it like this:

```
$ mkdir build
$ cd build
$ cmake -DCMAKE_BUILD_TYPE=DEBUG ..
$ make -j
```

The different logging levels are defined in `src/include/common/logger.h`. After enabling logging, the logging level default `LOG_LEVEL_INFO`. Any logging method with a level that is equal to or higher than `LOG_LEVEL_INFO` (e.g., `LOG_INFO`, `LOG_WA` `LOG_ERROR`) will emit logging information. Note that you will need to add `#include "common/logger.h"` to any file that yo want to use the logging infrastructure.

METHOD SIGNATURES

Please do **not** change any of the method signatures for any of the provided methods. You are free to add your own helper methods.

ADDING HELPER FUNCTIONS - TEMPLATING

If you are having trouble with function signatures when adding new functions, the following example may help.

Here is one of the important templates used in Project 2:

```
template <typename KeyType, typename ValueType, typename KeyComparator>
```

You can see that it does **not** appear directly above the `Remove` function in `extendible_hash_table.h`:

```
bool Remove(Transaction *transaction, const KeyType &key, const ValueType &value);
```

However, it is required directly above the `Remove` function in `extendible_hash_table.cpp`:

```
template <typename KeyType, typename ValueType, typename KeyComparator>
bool HASH_TABLE_TYPE::Remove(Transaction *transaction, const KeyType &key, const ValueType &value) {
  return false;
}
```

Also notice the `HASH_TABLE_TYPE` template on the `Remove` function. This is defined in `extendible_hash_table.h` as:

When adding a new function, the following steps should work:

Add function signature to the header file. Use private functions whenever possible. If you need to add a public function, be wary of name conflicts with the grading harness. You can preempt this by making your function names unique in some way there should typically not be conflicts.
Add the function to the `.cpp` file **with templates**. You will probably need a template before the function name like `TEMPLATE::FunctionName`. If you are using KeyType, ValueType, and/or KeyComparator, you will need the aforementioned template above the function signature.

Post all of your questions about this project on Piazza. Do not email the TAs directly with questior

GRADESCOPE FAILED TO EXECUTE

If Gradescope gives you an error about failing to execute, then the most likely cause is that your code doesn't compile or y logging invalid UTF-8 characters. Remove all logging, delete your local build folder, and rebuild your project.

GRADING RUBRIC

Each project submission will be graded based on the following criteria:

Does the submission successfully execute all of the test cases and produce the correct answer?
Does the submission execute without any memory leaks?

Note that we will use additional test cases that are more complex and go beyond the sample test cases that we provide yo

LATE POLICY

See the late policy in the syllabus.

SUBMISSION

After completing the assignment, you can submit your implementation of to Gradescope. Since we have two checkpoints f this project, you will need to submit them separately through the following link.

https://www.gradescope.com/courses/286490/

You only need to include the following files:

- `src/include/buffer/lru_replacer.h`
- `src/buffer/lru_replacer.cpp`
- `src/include/buffer/buffer_pool_manager_instance.h`
- `src/buffer/buffer_pool_manager_instance.cpp`
- `src/include/buffer/parallel_buffer_pool_manager.h`
- `src/buffer/parallel_buffer_pool_manager.cpp`
- `src/include/storage/page/hash_table_directory_page.h`
- `src/storage/page/hash_table_directory_page.cpp`
- `src/include/storage/page/hash_table_bucket_page.h`
- `src/storage/page/hash_table_bucket_page.cpp`
- `src/include/container/hash/extendible_probe_hash_table.h`
- `src/container/hash/extendible_probe_hash_table.cpp`

run the bash file to make things easier for you.

```
$ zip project2-submission.zip src/include/buffer/lru_replacer.h \
src/buffer/lru_replacer.cpp \
src/include/buffer/buffer_pool_manager_instance.h \
src/buffer/buffer_pool_manager_instance.cpp \
src/include/buffer/parallel_buffer_pool_manager.h \
src/buffer/parallel_buffer_pool_manager.cpp \
src/include/storage/page/hash_table_directory_page.h \
src/storage/page/hash_table_directory_page.cpp \
src/include/storage/page/hash_table_bucket_page.h \
src/storage/page/hash_table_bucket_page.cpp \
src/include/container/hash/extendible_hash_table.h \
src/container/hash/extendible_hash_table.cpp
```

You can submit your answers as many times as you like and get immediate feedback.

COLLABORATION POLICY

Every student has to work individually on this assignment.

Students are allowed to discuss high-level details about the project with others.

Students are **not** allowed to copy the contents of a white-board after a group meeting with other students.

Students are **not** allowed to copy the solutions from another colleague.

⚠ WARNING: All of the code for this project must be your own. You may not copy source code from other students or other sources that you find on the web. Plagiarism **will not** be tolerated. See CMU' **Policy on Academic Integrity** for additional information.

Last Updated: Jan 07,