p0-cs660

JiaHao He

February 2025

1 Explanation of `TODO pa0` Functions

1.1 Database Class

1.1.1 `Database::add(std::unique_ptr<DbFile> file)`

- Inserts a new `DbFile` into the database catalog, using `file->getName()` as the key.
- Throws `std::logic_error` if a file with the same name already exists.
- Moves the `unique_ptr<DbFile>` into `pImpl->files`, so the database officially owns it.

1.1.2 `Database::remove(const std::string& name)`

- Removes the file `name` from the catalog and returns its `unique_ptr<DbFile>` to the caller.
- Looks up `name` in `pImpl->files`; throws `std::logic_error` if it does not exist.
- Before erasing, calls `bufferPool.flushFile(name)` to ensure all dirty pages associated with this file are written to disk.
- Erases the key-value pair from `pImpl->files` and returns the `unique_ptr` via `std::move`.

1.1.3 `Database::get(const std::string& name) const`

- Retrieves a reference to the `DbFile` with the given name.
- Throws `std::logic_error` if `name` is not found in `pImpl->files`.
- Returns `*(it->second)`, the dereferenced pointer from the `unique_ptr<DbFile>`.

1.2 BufferPool Class

1.2.1 `BufferPool::BufferPool()`

- Initializes internal data structures, such as the array of frames, the LRU list, and the `pageTable`.
- All frames are initially marked *not in use* (free).

1.2.2 `BufferPool:: BufferPool()`

- Called when the `BufferPool` is destroyed.
- Iterates over frames; for each *in-use* and *dirty* frame, calls `flushOnePage` to ensure data is written back to disk.

1.2.3 `Page& BufferPool::getPage(const PageId& pid)`

- Returns a reference to the page corresponding to `pid`.
- Checks if `pid` is in the `pageTable`. If yes, moves that frame to the front of the LRU list (most recently used) and returns it.
- If not in memory, finds a free frame or evicts the LRU page if the cache is full. For an evicted page that is dirty, writes it back to disk first.
- Reads the requested page from disk into the frame (using `DbFile::readPage`), updates `pageTable`, and returns the page.

1.2.4 `void BufferPool::markDirty(const PageId& pid)`

- Sets the `dirty` flag for the page identified by `pid`.
- If `pid` is not in the buffer, it may either do nothing or throw an error (depending on design choice).

1.2.5 `bool BufferPool::isDirty(const PageId& pid) const`

- Returns `true` if the page is marked dirty.
- Throws an exception if `pid` is not present in the buffer pool (depending on the requirement to handle missing pages).

1.2.6 `bool BufferPool::contains(const PageId& pid) const`

- Returns whether `pid` is currently in the buffer pool.
- Typically a direct lookup in `pageTable`.

1.2.7 `void BufferPool::discardPage(const PageId& pid)`

- Removes the page from the buffer pool entirely, without writing it back to disk.
- This updates both the `pageTable` and the LRU list, and resets the frame to *free* status.

1.2.8 `void BufferPool::flushPage(const PageId& pid)`

- If the page with `pid` is dirty, writes it to disk via `DbFile::writePage`.
- Marks the frame as clean afterward.
- If `pid` is not in the pool, no action is taken or an exception may be thrown (implementation-dependent).

1.2.9 `void BufferPool::flushFile(const std::string& file)`

- Writes back all dirty pages whose `PageId` matches `file`.
- Invokes `flushPage` for each matching page.