Final Report: CSE 444

Name(s) & netID(s): Tasnim Alam (alamt) , Danno Muggli (dmuggli)

Section 1. Overall System Architecture (1.5 page)

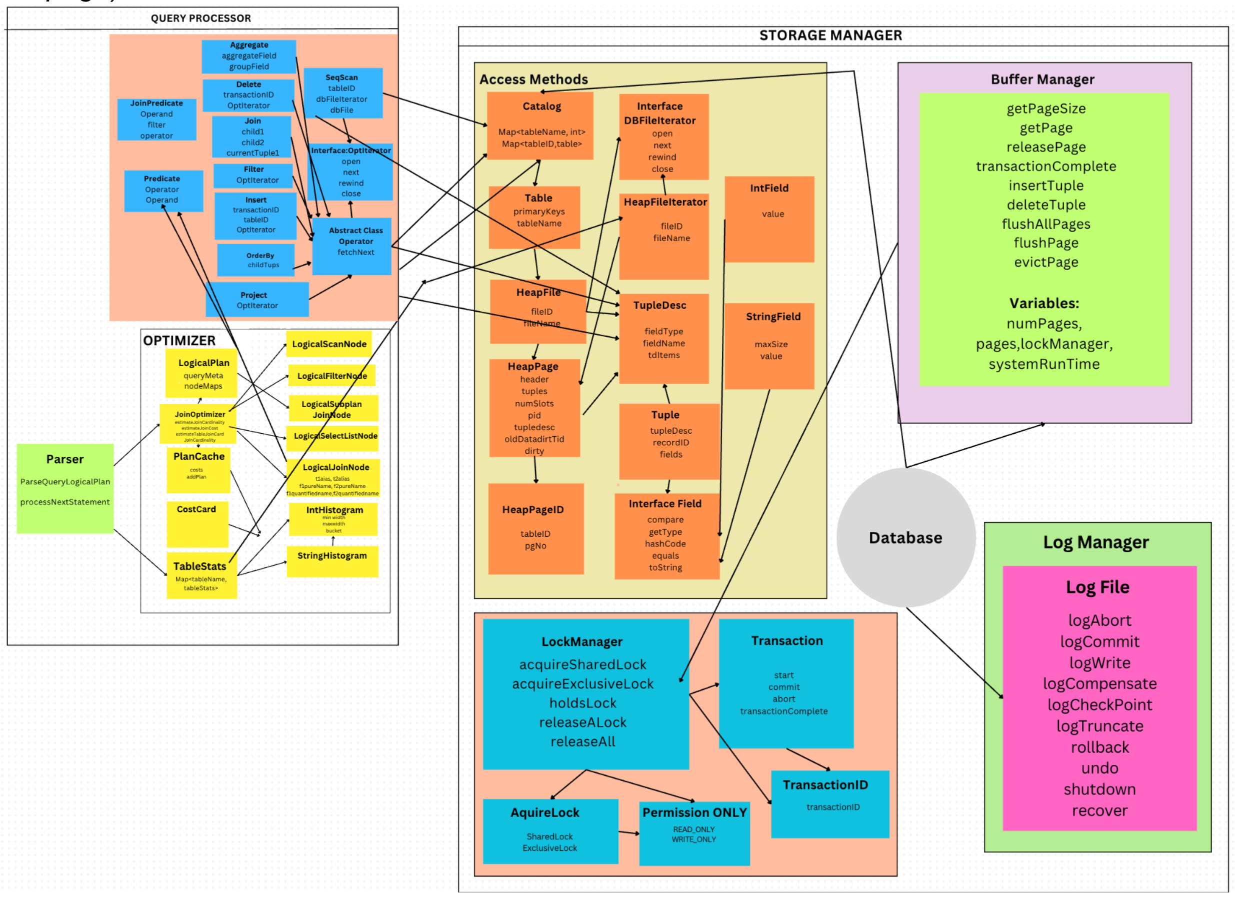
An overview description of the overall architecture of your final SimpleDB system (0.5 page)

Our version of SimpleDB has a modular architecture, the goal of that design is to keep the storage efficient and maximize the query execution capabilities. Our design consists of four main components: storage management, query execution, concurrency control and transactions, and query optimization.

The SimpleDB we implemented is an architecture that consists of several main components. One of the key components is the storage management module, responsible for handling data organization and retrieval on disk. This includes classes such as DbFile and HeapFile, which represent the physical file structure and individual pages of the database. The BufferPool class manages an in-memory buffer pool, allowing for SimpleDB to execute efficient caching and eviction policies that minimize disk I/O operations. Our query execution component is responsible for processing relational algebra operations and generating query results. This component consists of operators that implement specific operations like projection, join, and aggregation. These operators can be combined to form complex query plans, and the Query class handles the parsing and execution of SQL queries, generating the appropriate query plan and invoking the necessary operators. Concurrency control and transaction management are very important to ensure proper isolation and timing of concurrent transactions. The LockManager class handles the acquisition and release of locks on database objects to ensure isolation levels and prevent conflicts. The Transaction and TransactionId classes manage transaction state and identification. The LogFile and LogFileManager classes deal with transaction logging, enabling recovery in case of failures. Query optimization plays a crucial role in improving query performance. The optimizer uses techniques like cost estimation to select the most efficient execution plan for a given query. Factors like indexes, statistics, and selectivity are considered during the optimization process to minimize query execution time.

Our SimpleDB architecture allows for a modular approach among these components. Storage management ensures efficient data handling, query execution handles processing and optimization, concurrency control manages transactions, and query optimization improves performance. Our modular design enables customization. This makes our version of SimpleDB flexible and scalable.

A diagram of the overall system architecture. You may modify our diagram from the early slides. (0.5 page)



Note:

1. You might have to zoom in to see the writings inside the boxes, apologizing for that in advance.
2. We skipped Process Manager and Shared Utilities because these topics were not covered for the implementation of our simpleDB.

A somewhat more detailed description of each of the following main components of your SimpleDB system: Buffer manager, operators, lock manager, and log manager. Briefly explain how each work and any major choices you made in implementation. (0.5 page)

For our implementation, the buffer manager is responsible for managing the buffer pool. The buffer pool is a portion of the main memory that is used to cache database pages. The goal of the buffer pool is to minimize disk I/O by keeping the frequently accessed pages in memory. The buffer manager is mainly responsible for the following functions: fetching pages, replacing pages, flushing pages, and concurrency controls.

When a page is requested, the buffer manager will perform a check to see if the page is already in the buffer pool. If the requested page is not in the buffer pool, it will fetch the page from disk and insert it into the buffer pool. This leads us to the next portion of the implementation, page replacement. If the buffer pool is full, the buffer manager will use the page replacement policy to select a page to evict. This makes space for the page that was just requested to enter into the buffer pool. The buffer manager is also tasked with maintaining page flushing. After a page has been modified and it needs to be written back to disk, the buffer manager will handle the process to ensure durability and consistency. The final component is the concurrency control. This is when the buffer manager coordinates with the lock manager to ensure that synchronization is maintained.

In general, operators are responsible for executing the relational algebra operations and producing the query results. project, join, and aggregator. The main purpose of all these operators is that they will take in tuples and generate output tuples based on the specified operations. The operators themselves are simplistic, but when they are nested they can produce complex query plans.

The log manager is tasked with handling transaction logging and recovery. It will ensure the durability and atomicity of transactions by writing log records that capture changes made by transactions before modifying the actual database.The log manager is performing functions such as, logging, checkpointing, and recovery.

Section 2. Discussion (0.5-1 page)

Discuss the overall performance of your SimpleDB engine.

We implemented SimpleDB by using the timeout policy as opposed to using the dependency graph. Our approach set a time limit for transactions to complete. If the time was exceeded the transaction was considered to have either failed or aborted. By using the timeout policy we take out of consideration the dependencies of the specific transactions and the operations that are being performed. This approach is more simplistic than the dependency graph, but for our purposes it worked well.

The main issue with using the timeout policy is that it opened our implementation up to having unnecessary aborts. The efficiency of SimpleDB can be impacted because the timeout may be due to legitimate reasons. A timeout may occur because there is a heavy workload at the time. This can abort the transaction only to have it redo the transaction a short time later. This re-execution of transactions can adversely impact the overall speed and efficiency of SimpleDB.

Using the timeout policy is more simplistic because it also does not allow for precise tracking of dependencies and serialization of the transactions. We attempted to implement the dependency graph, but since we couldn’t get it quite right in the time allowed for the assignment, we stuck with the timeout policy. We know that the dependency graph would offer better concurrency control, avoid conflicts more easily, and maintain better data consistency. However, we still feel that SimpleDB is able to perform the duties expected of most DBMS.

Discuss what you would implement or what you would change in the implementation if you had more time.

We think a great way to improve our version of SimpleDB would be to add more query statistics collection capability. We could extend the TableStats class to gather and maintain stats about the data distribution that is in the table. We would need to collect things like the number of tuples and distinct values. These stats could be used by the query optimizer to come up with more accurate query plans. This would increase performance.

We could iterate through the table to collect the number of tuples and maintain the data structures to monitor distinct attribute values. The query optimizer could leverage this data and statistics to aid in estimations about selectivity and cardinality. This will allow for more informed decision making during the optimization process. It could improve things like join order and index selection. Overall, the addition of greater query statistics collection and maintenance will enhance the efficiency of the query execution process of our SimpleDB.