Lelio GUALINO, Youri HALMAERT, Estelle CADENE

Lab 3 - SimpleDB Writeup

Design Decisions :

Page Eviction Policy We have implemented the LRU page eviction policy. The policy of LRU guarantees that the least recently accessed page will be the first to be evicted once the buffer pool exceeds its capacity. This strategy has a nice balance between simplicity and efficiency: it doesn't retain stale pages, but it still manages to keep the most recent data in memory. For implementing LRU, a HashMap was used in addition to a doubly linked list, enabling efficient tracking of page usage.

Pros : LRU is simple to implement and hence widely deployed.

It works well for workloads that show temporal locality: pages accessed frequently are likely to be reused soon.

Tradeoffs : LRU requires extra bookkeeping to track usage, which slightly increases the overhead.

For workloads without temporal locality, LRU may evict pages that will be accessed again soon.

Join Implementation

We have implemented the Join operator using a nested-loops join. The algorithm is straightforward but far from optimal for large relations.

Tradeoffs: Pros: The algorithm is pretty easy to implement. Cons: The performance may be terrible for large inputs due to its complexity.

Given the scope of this lab, we decided not to implement more sophisticated in-memory join algorithms, such as hash joins or sort-merge joins.

API Changes

No significant modifications were made to the API provided. The existing abstractions, such as OpIterator, BufferPool, and HeapPage, were enough to implement the necessary functionality. Minor utility methods were added for internal use, such as updating the LRU metadata in BufferPool and helper functions for correctness testing.

Missing or Incomplete Elements

The implementation implements all the required functionality and passes all the mentioned unit and system tests. However, we have not implemented advanced join algorithms.

Challenges Overcome

1. Aggregates with Grouping

Handling the aggregates with grouping was very tricky as it was very important to correctly map every tuple to the appropriate group. The most time-consuming part was to implement the Aggregator interface using appropriate data structures - HashMap for maintaining group values.

We iteratively merged tuples into groups using HashMap entries, making sure that COUNT, SUM, AVG, MIN, and MAX were computed correctly.

2. Buffer Pool Eviction

Instrumenting the eviction logic to work seamlessly with the BufferPool and comply with test requirements, such as the difference between flushPage vs. evictPage was initially confusing.

Solution:

We made clear the difference between flushing pages - which means writing dirty pages to disk - and evicting pages, which means removing them from memory by separating these responsibilities into different methods.

Time Spent

Total Time: Approximately around 13 hours. Breakdown:

- Reading and understanding the lab documentation: 2 hours.
- Implementing individual components (Filter, Join, Aggregate): 6 hours.
- Testing and debugging: 4 hours.
- Writeup preparation and final review: 1 hour.

Difficulties Encountered

Conceptual Overhead: To understand how different SimpleDB components interact, such as BufferPool, HeapPage, and operators, it took some time.

Testing: When tests failed with uninformative messages, it was often necessary to step through code in the debugger to understand how edge cases-like handling empty slots or evicting dirty pages-should be handled.

Conclusion

This laboratory was a very good exercise to realize main features of any database. It gave a chance to go deeper into query processing and buffer pool management and to realize the complexity of relational database systems. Although a few components could be implemented more efficiently-for example, some join algorithms-the lab's objectives were accomplished.