50.043 Database Lab 2 Report

Name	Student ID
Matthaeus Choo Zhi Jie (Zhu Zhijie)	1008103
Lee Jing Kang	1008240
Chong Juo Ru Faustina	1007967

50.043 Database System and Big Data ~ Lab 2

Exercise 1 ~ Filter and Join

The Predicate and JoinPredicate classes contai logic for comparing tuple fields. These are for filtering and joining records based on specified conditions. Predicate handles single-tuple comparisons, while JoinPredicate applies to two-tuples from different relations. Both rely on the Field.compare() method for evaluating relational operations like equality or inequality.For the Filter operator, we implemented a standard selection mechanism. It iterates over tuples from its child operator and returns those that satisfy the predicate. This forms the basis for implementing SQL WHERE clauses. The most difficult part of the exercise was implementing the Join operator. We used a nested loops join strategy, where for each tuple in the outer relation, we iterate over all tuples in the inner relation. If a pair of tuples satisfies the join predicate, we concatenate their fields into a new joined tuple. The implementation supports any binary predicate, and the resulting output includes both input tuple descriptors merged together.

Exercise 2 ~ Aggregates

The functions that have been implemented into classes StringAggregator, IntegerAggregator, and Aggregate are there to support aggregation functions such as COUNT, SUM, AVG, MIN and MAX for string and integer tuples. Both hashmaps maintain a count hashmap to support counting operations, with IntegerAggregator maintaining another to support running computations such as sums and min/max values. Both String and Integer functions use gbfield with logics inside the relevant functions to handle grouping types.

For the aggregate class open function, field type checks are added to prevent invalid operations being performed. A function aggname has been added to return the string version of the aggregate operation to make the code more readable.

Exercise 3 ~ Heapfile and HeapPage

The implementation of insertTuple and deleteTuple in HeapFile and HeapPage helps in efficient data maninpulation of the database's heap structure. At the page level, HeapPage utilizes a bitmap header structure to track slot occupancy, providing an efficient O(1) mechanism for identifying available tuple slots while minimizing storage overhead. This design is complemented by lazy tuple loading, which enhances performance by only materializing tuples when accessed. The class maintains important invariants through immutable fields (HeapPageId, TupleDesc) and supports transaction recovery through a before-image mechanism that preserves previous page states.

At the file abstraction level, HeapFile implements a page-granular storage manager that intelligently interacts with the buffer pool to optimize I/O operations. The implementation exhibits several notable features: dynamic file growth through append-only page allocation, transaction-aware page access through proper permission management, and an iterator implementation that employs lazy page loading to minimize memory consumption during full table scans. Both classes incorporate error handling to maintain data integrity, including validation checks for tuple descriptor compatibility and proper slot state management. The design effectively balances several competing concerns: space efficiency through compact metadata representation, performance through minimized I/O operations, and correctness through transaction isolation.

Exercise 4 ~ Insertion and Deletion

The Insert operator reads tuples from its child operator and inserts them into the specified table using the BufferPool.insertTuple() method. It ensures schema compatability by verifying that the child's TupleDesc matches the target table's schema before insertion. The operator returns count of inserted records in a single tuple and maintains its state with a hasInserted flag to prevent multiple outputs. Error handling catches failures during insertion (invalid table ID, or I/O errors).
The Delete operator processes tuples from child operator and removes them by BufferPool.deleteTuple(). Likewise, it outputs the count of deleted records in a single tuple and has a flag hasDeleted to ensure 1 return value.
Both operators manage resource lifecycle using the open(), close() and rewind() methods. This implementation assumes tuples exist in the database and delegates concurrency control to BufferPool.

Exercise 5 ~ Page eviction

For page eviction in the buffer pool, we implemented the evictPage() method in BufferPool.java. The key requirement was to ensure that dirty pages are never evicted. Our implementation iterates through the cached pages to find a clean page (i.e., one where isDirty() == null). Once found, the page is flushed to disk using flushPage() and removed from the buffer pool. If all pages are dirty, a DbException is thrown to indicate that eviction is not possible. This prevents potential data loss due to unflushed dirty pages being discarded.

Missing/Incomplete Elements Of Our Code/Changes Made To The API

All Lab 1 and Lab 2 code have been implemented. No Changes made to the API.