LAB3

1 exer1

1. `Predicate` class:
   The `Predicate` class represents a predicate that compares tuples to a specified field value using a given operator (`EQUALS`, `GREATER_THAN`, `LESS_THAN`, etc.). It has methods to get the field number, operator, and operand, as well as a `filter` method that checks if a given tuple satisfies the predicate.
2. `JoinPredicate` class:
   The `JoinPredicate` class represents a predicate that compares fields of two tuples using a given operator. It has a constructor that takes two field indices and an operator, as well as a `filter` method that checks if two tuples satisfy the join predicate.
3. `Filter` operator:
   The `Filter` operator implements a relational select operation. It takes a `Predicate` and a child `DbIterator` (which provides tuples to filter). The `fetchNext` method iterates over the child iterator, applying the predicate to each tuple and returning those that pass the filter.
4. `Join` operator:
   The `Join` operator implements the relational join operation. It takes a `JoinPredicate` and two child `DbIterator` instances (representing the left and right relations to join). The `fetchNext` method implements a nested loops join algorithm, iterating over the left relation and, for each tuple, checking if any tuples from the right relation satisfy the join predicate. If a match is found, the method returns the concatenation of the two tuples.
5. `HashEquiJoin` operator:
   The `HashEquiJoin` operator is an implementation of the hash join algorithm for equi-joins (joins where the predicate is an equality comparison). It first builds a hash table from the right relation, using the join attribute as the key. Then, for each tuple in the left relation, it probes the hash table to find matching tuples from the right relation and returns the concatenated tuples.

2 exer2

1. `IntegerAggregator` class:

   - This class knows how to compute aggregate operations (MIN, MAX, SUM, AVG, COUNT) over a set of `IntField` values.
   - It maintains a `HashMap` to store the aggregate values for each group (or a single value if no grouping is specified).
   - The `mergeTupleIntoGroup` method updates the aggregate values based on the incoming tuple.
   - The `iterator` method returns a `DbIterator` over the computed aggregate results.

2. `StringAggregator` class:

   - This class knows how to compute the COUNT aggregate operation over a set of `StringField` values.
   - It maintains a `HashMap` to store the count of tuples for each group (or a single count if no grouping is specified).
   - The `mergeTupleIntoGroup` method updates the counts based on the incoming tuple.
   - The `iterator` method returns a `DbIterator` over the computed aggregate counts.

3. `Aggregate` operator:

   - This operator computes an aggregate (sum, avg, max, min) over a single column, optionally grouped by another column.
   - The constructor takes the child `DbIterator` (which provides the tuples), the index of the aggregate field, the index of the grouping field (or -1 for no grouping), and the aggregate operation to perform.
   - Depending on the type of the aggregate field (`IntField` or `StringField`), it creates an instance of either `IntegerAggregator` or `StringAggregator`.
   - The `open` method iterates over the child iterator, merging each tuple into the appropriate group in the aggregator.
   - The `fetchNext` method returns the next tuple from the aggregator's iterator, containing the group value and the aggregate result.

- The `getTupleDesc` method constructs the `TupleDesc` for the output tuples, including the group field (if present) and the aggregate field with an informative name.
- The `close` method closes the child iterator and the aggregator's iterator.

3 exer3

1. `HeapFile` class:

   - Represents a file on disk that stores tuples (rows) in no particular order.
   - Provides methods to read and write pages from/to disk, insert and delete tuples, and create an iterator over the tuples.
   - Manages the mapping between table IDs, page IDs, and the actual file on disk.
   - Handles the creation of new pages when inserting tuples and there is no available space on existing pages.

2. `HeapPage` class:

   - Represents a single page in a `HeapFile`.
   - Stores the page header, which indicates which slots on the page are in use, and the actual tuple data.
   - Provides methods to read and write tuples to/from the page, mark slots as used or unused, and create an iterator over the tuples on the page.
   - Handles serialization and deserialization of page data to/from byte arrays for storage on disk.

4 exer4

1. `Insert` operator:

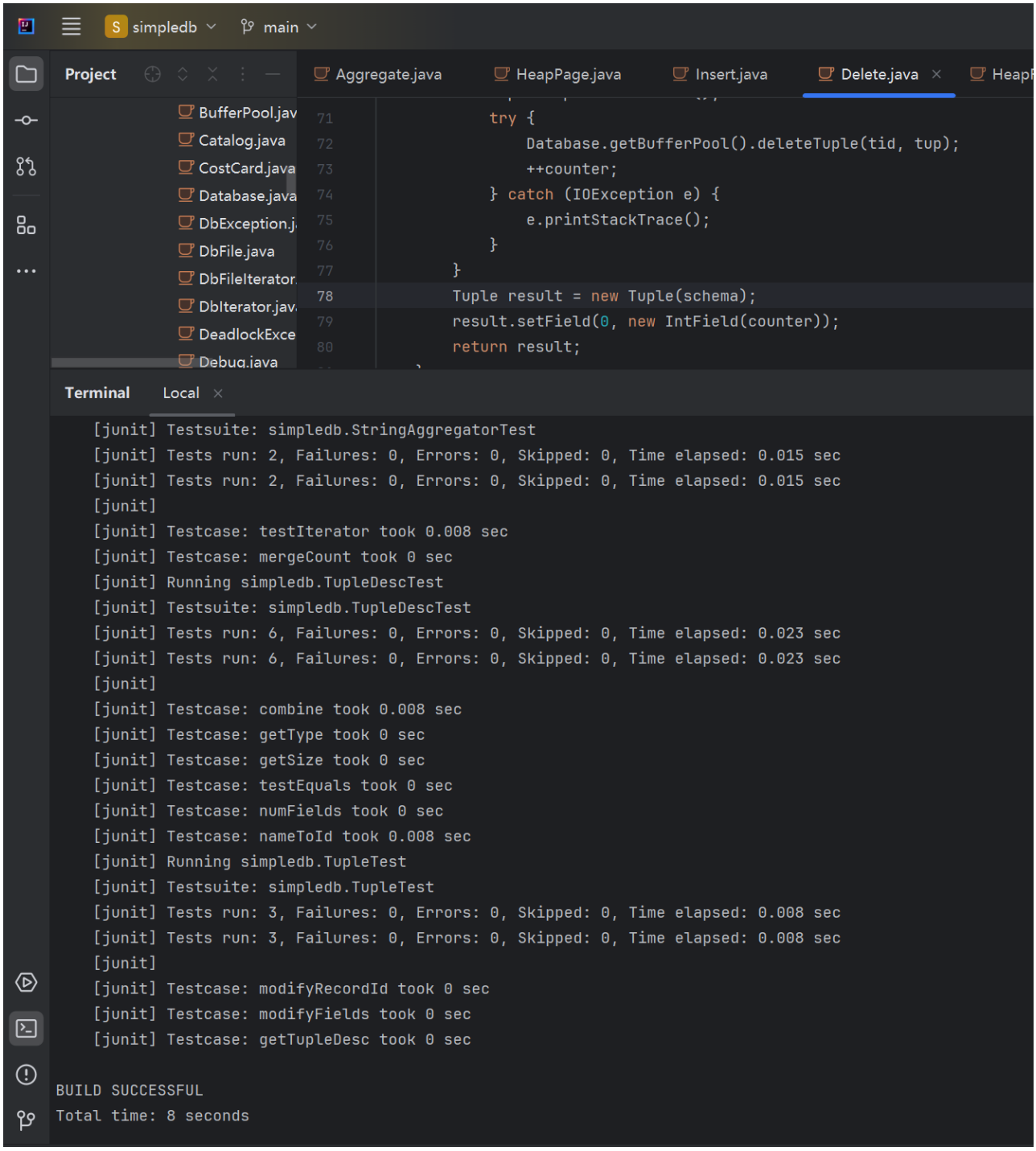
   - This operator is responsible for inserting tuples (rows) into a specified table.
   - The constructor takes a `TransactionId`, a child `DbIterator` (which provides the tuples to be inserted), and the table ID to insert into.
   - The `fetchNext` method reads tuples from the child iterator and inserts them into the table using the `BufferPool.insertTuple` method.
   - It returns a single-field tuple containing the number of inserted records.
   - If `fetchNext` is called more than once, it returns `null`.

2. `Delete` operator:

   - This operator is responsible for deleting tuples from a table.
   - The constructor takes a `TransactionId` and a child `DbIterator` (which provides the tuples to be deleted).
   - The `fetchNext` method reads tuples from the child iterator and deletes them from the table using the `BufferPool.deleteTuple` method.
   - It returns a single-field tuple containing the number of deleted records.
   - If `fetchNext` is called more than once, it returns `null`.

5 result

ant test:



ant systemtest:

simpledb main

Project Aggregate.java HeapPage.java Insert.java Delete.java He

Terminal Local

```
    [junit] Running simpledb.systemtest.HashEquiJoinTest
    [junit] Testsuite: simpledb.systemtest.HashEquiJoinTest
    [junit] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.189 sec
    [junit] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.189 sec
    [junit]
    [junit] Testcase: testNoMatch took 0.173 sec
    [junit] Testcase: testSingleMatch took 0 sec
    [junit] Testcase: testMultipleMatch took 0.008 sec
    [junit] Running simpledb.systemtest.InsertTest
    [junit] Testsuite: simpledb.systemtest.InsertTest
    [junit] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.158 sec
    [junit] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.158 sec
    [junit]
    [junit] Testcase: testEmptyToEmpty took 0.141 sec
    [junit] Testcase: testOneToOne took 0.002 sec
    [junit] Testcase: testOneToEmpty took 0 sec
    [junit] Testcase: testEmptyToOne took 0.007 sec
    [junit] Running simpledb.systemtest.JoinTest
    [junit] Testsuite: simpledb.systemtest.JoinTest
    [junit] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.153 sec
    [junit] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.153 sec
    [junit]
    [junit] Testcase: testMultipleMatch took 0.137 sec
    [junit] Testcase: testNoMatch took 0 sec
    [junit] Testcase: testSingleMatch took 0.008 sec
    [junit] Running simpledb.systemtest.ScanTest
    [junit] Testsuite: simpledb.systemtest.ScanTest
    [junit] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.424 sec
    [junit] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.424 sec
    [junit]
    [junit] Testcase: testRewind took 0.18 sec
    [junit] Testcase: testSmall took 0.154 sec
    [junit] Testcase: testCache took 0.09 sec

BUILD SUCCESSFUL
Total time: 11 seconds
PS D:\G\大三下\数据库原理\github_repository\acmdblab\lab3>
```