LAB4

521120910245 严冬

1 structure

1. `IntHistogram`: This class represents a fixed-width histogram over a single integer-based field. The main functionality is to create a histogram for an integer field, add values to the histogram, and estimate the selectivity of a given predicate on that field.
2. `TableStats`: This class maintains statistics (e.g., histograms) about base tables in a query. It creates histograms for all fields in a table and provides methods to estimate costs and cardinalities of various operations, such as scanning a table or performing a join.
3. `JoinOptimizer`: This class is responsible for ordering a series of joins optimally and selecting the best instantiation of a join for a given logical plan. The key method is `orderJoins`, which takes table statistics and filter selectivities as input and returns an optimal order of joins using a dynamic programming approach.

The main implementation in the provided code is as follows:

1. In the `IntHistogram` class:

   - The constructor initializes the histogram with the specified number of buckets, minimum and maximum values.
   - The `addValue` method adds a value to the histogram.
   - The `estimateSelectivity` method estimates the selectivity of a given predicate (e.g., `=`, `<`, `>`) and operand on the histogram.

2. In the `TableStats` class:

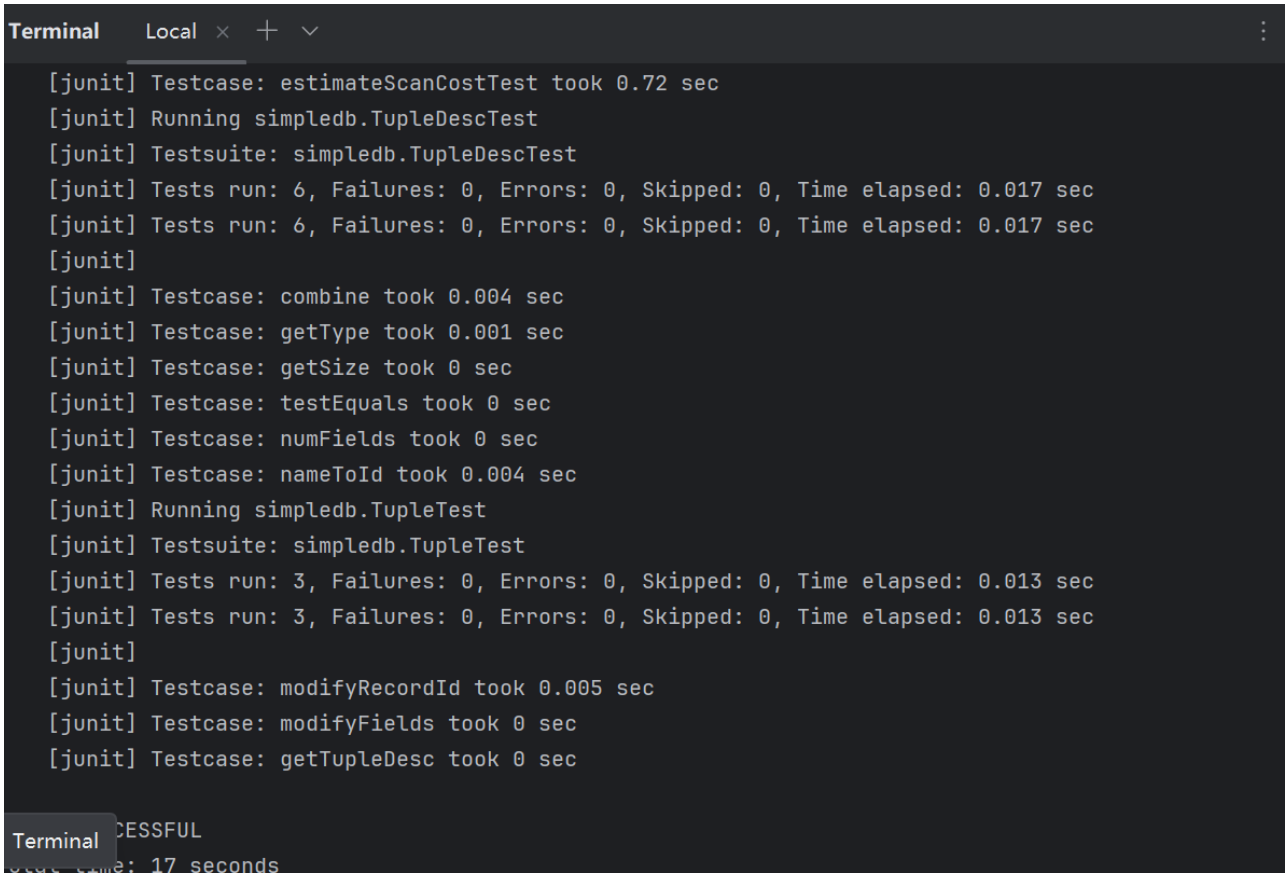
   - The constructor scans the table's data and creates histograms for all integer and string fields.
   - The `estimateScanCost` method estimates the cost of sequentially scanning the table.
   - The `estimateTableCardinality` method estimates the number of tuples in the relation after applying a given selectivity factor.
   - The `estimateSelectivity` method estimates the selectivity of a predicate on a specific field using the corresponding histogram.

3. In the `JoinOptimizer` class:

   - The `orderJoins` method is the main algorithm for finding the optimal join order using dynamic programming. It enumerates all subsets of joins and computes the cost and cardinality of each subset using the `computeCostAndCardOfSubplan` method. It maintains a cache (`PlanCache`) to avoid recomputing subplans.
   - The `computeCostAndCardOfSubplan` method recursively computes the cost and cardinality of joining a new join node to an existing subset of joins.
   - The `estimateJoinCost` and `estimateJoinCardinality` methods estimate the cost and cardinality of a join operation, respectively, based on the join algorithm used.

2 test

all pass




```
    [junit] Added scan of table emp
    [junit] Added scan of table dept
    [junit] Added scan of table hobbies
    [junit] Added scan of table hobby
    [junit] Added join between emp.c1 and dept.c0
    [junit] Added join between hobbies.c0 and emp.c2
    [junit] Added join between hobbies.c1 and hobby.c0
    [junit] Added select list field null.*
    [junit] Transaction 5 aborted because of unhandled error
    [junit] Invalid SQL expression:
    [junit]     simpledb.ParsingException: Unknown field in filter expression emp.c3
    [junit] ------------ --------------- --------------
    [junit]
    [junit] Testcase: queryTest took 1.244 sec
    [junit] Running simpledb.systemtest.ScanTest
    [junit] Testsuite: simpledb.systemtest.ScanTest
    [junit] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.567 sec
    [junit] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.567 sec
    [junit]
    [junit] Testcase: testRewind took 0.203 sec
    [junit] Testcase: testSmall took 0.246 sec
    [junit] Testcase: testCache took 0.109 sec

BUILD SUCCESSFUL
Total time: 14 seconds
```