

## Join GitHub today

GitHub is home to over 31 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)

Branch: master ▾ Fa18HW5 / src / main / java / edu / berkeley / cs186 / database / query / IndexScanOperator.java

[Find file](#) [Copy path](#)

es1024 Fixed testQueryWithIndex/testSortedScanNoIndexLocking failing with hw...

c35db97 on Nov 11, 2018

1 contributor

240 lines (216 sloc) 9.35 KB

[Raw](#) [Blame](#) [History](#)   

```
1 package edu.berkeley.cs186.database.query;
2
3 import edu.berkeley.cs186.database.Database;
4 import edu.berkeley.cs186.database.DatabaseException;
5 import edu.berkeley.cs186.database.concurrency.LockType;
6 import edu.berkeley.cs186.database.concurrency.LockUtil;
7 import edu.berkeley.cs186.database.databox.DataBox;
8 import edu.berkeley.cs186.database.table.Record;
9 import edu.berkeley.cs186.database.table.Schema;
10 import edu.berkeley.cs186.database.table.stats.TableStats;
11 import edu.berkeley.cs186.database.table.stats.Histogram;
12
13 import java.util.ArrayList;
14 import java.util.Iterator;
15 import java.util.List;
16 import java.util.NoSuchElementException;
17
18 public class IndexScanOperator extends QueryOperator {
19     private Database.Transaction transaction;
20     private String tableName;
21     private String columnName;
22     private QueryPlan.PredicateOperator predicate;
23     private DataBox value;
24
25     private int columnIndex;
26
27     /**
28      * An index scan operator.
29      *
30      * @param transaction the transaction containing this operator
31      * @param tableName the table to iterate over
32      * @param columnName the name of the column the index is on
33      * @throws QueryPlanException
34      * @throws DatabaseException
35      */
36     public IndexScanOperator(Database.Transaction transaction,
37                             String tableName,
38                             String columnName,
39                             QueryPlan.PredicateOperator predicate,
40                             DataBox value) throws QueryPlanException, DatabaseException {
41         super(OperatorType.INDEXSCAN);
42         this.tableName = tableName;
43         this.transaction = transaction;
```

```

44     this.columnName = columnName;
45     this.predicate = predicate;
46     this.value = value;
47     this.setOutputSchema(this.computeSchema());
48     columnName = this.checkSchemaForColumn(this.getOutputSchema(), columnName);
49     this.columnIndex = this.getOutputSchema().getFieldNames().indexOf(columnName);
50
51     /*
52     this.stats = this.estimateStats();
53     this.cost = this.estimateIOCost();
54     */
55 }
56
57 public String str() {
58     return "type: " + this.getType() +
59         "\ntable: " + this.tableName +
60         "\ncolumn: " + this.columnName +
61         "\nopoperator: " + this.predicate +
62         "\nvalue: " + this.value;
63 }
64
65 /**
66  * Returns the column name that the index scan is on
67  *
68  * @return columnName
69  */
70 public String getColumnName() {
71     return this.columnName;
72 }
73
74 /**
75  * Estimates the table statistics for the result of executing this query operator.
76  *
77  * @return estimated TableStats
78  */
79 public TableStats estimateStats() throws QueryPlanException {
80     TableStats stats;
81
82     try {
83         stats = this.transaction.getStats(this.tableName);
84     } catch (DatabaseException de) {
85         throw new QueryPlanException(de);
86     }
87
88     return stats.copyWithPredicate(this.columnIndex,
89                                     this.predicate,
90                                     this.value);
91 }
92
93 /**
94  * Estimates the IO cost of executing this query operator.
95  * You should calculate this estimate cost with the formula
96  * taught to you in class. Note that the index you've implemented
97  * in this project is an unclustered index.
98  *
99  * You will find the following instance variables helpful:
100  * this.transaction, this.tableName, this.columnName,
101  * this.columnIndex, this.predicate, and this.value.
102  *
103  * You will find the following methods helpful: this.transaction.getStats,
104  * this.transaction.getNumRecords, this.transaction.getNumIndexPages,
105  * and tableStats.getReductionFactor.
106  *
107  * @return estimate IO cost
108  * @throws QueryPlanException
109  */
110 public int estimateIOCost() throws QueryPlanException {

```

```

102 long numRecords;
103 long numIndexPages;
104 TableStats tableStats;
105 try {
106     numRecords = this.transaction.getNumRecords(this.tableName);
107     numIndexPages = this.transaction.getNumIndexPages(this.tableName, this.columnName);
108     tableStats = this.transaction.getStats(this.tableName);
109
110 } catch (DatabaseException err) {
111     throw new QueryPlanException("Can't find the number of records in IndexScanOperator#estimateIOCost().");
112 }
113
114 return (int)(tableStats.getHistograms().get(columnIndex).getCount() +
115             numIndexPages); //round up and cast to an int
116 }
117
118 public Iterator<Record> iterator() throws QueryPlanException, DatabaseException {
119     return new IndexScanIterator();
120 }
121
122 public Schema computeSchema() throws QueryPlanException {
123     try {
124         return this.transaction.getFullyQualifiedSchema(this.tableName);
125     } catch (DatabaseException de) {
126         throw new QueryPlanException(de);
127     }
128 }
129
130 /**
131  * An implementation of Iterator that provides an iterator interface for this operator.
132  */
133 private class IndexScanIterator implements Iterator<Record> {
134     private Iterator<Record> sourceIterator;
135     private Record nextRecord;
136
137     public IndexScanIterator() throws QueryPlanException, DatabaseException {
138         this.nextRecord = null;
139         if (IndexScanOperator.this.predicate == QueryPlan.PredicateOperator.EQUALS) {
140             this.sourceIterator = IndexScanOperator.this.transaction.lookupKey(
141                 IndexScanOperator.this.tableName,
142                 IndexScanOperator.this.columnName,
143                 IndexScanOperator.this.value);
144         } else if (IndexScanOperator.this.predicate == QueryPlan.PredicateOperator.LESS_THAN ||
145                 IndexScanOperator.this.predicate == QueryPlan.PredicateOperator.LESS_THAN_EQUALS) {
146             this.sourceIterator = IndexScanOperator.this.transaction.sortedScan(
147                 IndexScanOperator.this.tableName,
148                 IndexScanOperator.this.columnName);
149         } else if (IndexScanOperator.this.predicate == QueryPlan.PredicateOperator.GREATER_THAN) {
150             this.sourceIterator = IndexScanOperator.this.transaction.sortedScanFrom(
151                 IndexScanOperator.this.tableName,
152                 IndexScanOperator.this.columnName,
153                 IndexScanOperator.this.value);
154             while (this.sourceIterator.hasNext()) {
155                 Record r = this.sourceIterator.next();
156
157                 if (r.getValues().get(IndexScanOperator.this.columnIndex)
158                     .compareTo(IndexScanOperator.this.value) > 0) {
159                     this.nextRecord = r;
160                     break;
161                 }
162             }
163         } else if (IndexScanOperator.this.predicate == QueryPlan.PredicateOperator.GREATER_THAN_EQUALS) {
164             this.sourceIterator = IndexScanOperator.this.transaction.sortedScanFrom(
165                 IndexScanOperator.this.tableName,
166                 IndexScanOperator.this.columnName,
167                 IndexScanOperator.this.value);

```

```

178     }
179 }
180
181 /**
182  * Checks if there are more record(s) to yield
183  *
184  * @return true if this iterator has another record to yield, otherwise false
185  */
186 public boolean hasNext() {
187     if (this.nextRecord != null) {
188         return true;
189     }
190     if (IndexScanOperator.this.predicate == QueryPlan.PredicateOperator.LESS_THAN) {
191         if (this.sourceIterator.hasNext()) {
192             Record r = this.sourceIterator.next();
193             if (r.getValues().get(IndexScanOperator.this.columnIndex)
194                 .compareTo(IndexScanOperator.this.value) >= 0) {
195                 return false;
196             }
197             this.nextRecord = r;
198             return true;
199         }
200         return false;
201     } else if (IndexScanOperator.this.predicate == QueryPlan.PredicateOperator.LESS_THAN_EQUALS) {
202         if (this.sourceIterator.hasNext()) {
203             Record r = this.sourceIterator.next();
204             if (r.getValues().get(IndexScanOperator.this.columnIndex)
205                 .compareTo(IndexScanOperator.this.value) > 0) {
206                 return false;
207             }
208             this.nextRecord = r;
209             return true;
210         }
211         return false;
212     }
213     if (this.sourceIterator.hasNext()) {
214         this.nextRecord = this.sourceIterator.next();
215         return true;
216     }
217     return false;
218 }
219
220 /**
221  * Yields the next record of this iterator.
222  *
223  * @return the next Record
224  * @throws NoSuchElementException if there are no more Records to yield
225  */
226 public Record next() {
227     if (this.hasNext()) {
228         Record r = this.nextRecord;
229         this.nextRecord = null;
230         return r;
231     }
232     throw new NoSuchElementException();
233 }
234
235 public void remove() {
236     throw new UnsupportedOperationException();
237 }
238 }
239 }

```