JOIN OPERATOR

```java
package query;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;


import common.BacktrackingIterator;
import database.Database;
import database.DatabaseException;
import databox.DataBox;
import databox.Type;
import io.Page;
import table.Record;
import table.RecordId;
import table.RecordIterator;
import table.Schema;

public abstract class JoinOperator extends QueryOperator {

  public enum JoinType {
    SNLJ,
    PNLJ,
    BNLJ,
    INLJ,
    GRACEHASH,
    SORTMERGE
  }

  private JoinType joinType;
  private QueryOperator leftSource;
  private QueryOperator rightSource;
  private int leftColumnIndex;
  private int rightColumnIndex;
  private String leftColumnName;
  private String rightColumnName;
  private Database.Transaction transaction;

  /**
   * Create a join operator that pulls tuples from leftSource and rightSource.
   Returns tuples for which
   * leftColumnName and rightColumnName are equal.
   *
   * @param leftSource the left source operator
   * @param rightSource the right source operator
   * @param leftColumnName the column to join on from leftSource
   * @param rightColumnName the column to join on from rightSource
   * @throws QueryPlanException
   */
  public JoinOperator(QueryOperator leftSource,
                      QueryOperator rightSource,
                      String leftColumnName,
                      String rightColumnName,
                      Database.Transaction transaction,
                      JoinType joinType) throws QueryPlanException {
    super(OperatorType.JOIN);
    this.joinType = joinType;
    this.leftSource = leftSource;
    this.rightSource = rightSource;
    this.leftColumnName = leftColumnName;
    this.rightColumnName = rightColumnName;
    this.setOutputSchema(this.computeSchema());
    this.transaction = transaction;
  }

  public abstract Iterator<Record> iterator() throws QueryPlanException,
  DatabaseException;
```

```
66
67      @Override
68      public QueryOperator getSource() throws QueryPlanException {
69        throw new QueryPlanException("There is no single source for join operators.
          Please use " +
70            "getRightSource and getLeftSource and the corresponding set methods.");
71      }
72
73      public QueryOperator getLeftSource() {
74        return this.leftSource;
75      }
76
77      public QueryOperator getRightSource() {
78        return this.rightSource;
79      }
80
81      public void setLeftSource(QueryOperator leftSource) {
82        this.leftSource = leftSource;
83      }
84
85      public void setRightSource(QueryOperator rightSource) {
86        this.rightSource = rightSource;
87      }
88
89      public Schema computeSchema() throws QueryPlanException {
90        Schema leftSchema = this.leftSource.getOutputSchema();
91        Schema rightSchema = this.rightSource.getOutputSchema();
92        List<String> leftSchemaNames = new ArrayList<String>(leftSchema.getFieldNames());
93        List<String> rightSchemaNames = new
          ArrayList<String>(rightSchema.getFieldNames());
94        this.leftColumnName = this.checkSchemaForColumn(leftSchema, this.leftColumnName);
95        this.leftColumnIndex = leftSchemaNames.indexOf(leftColumnName);
96        this.rightColumnName = this.checkSchemaForColumn(rightSchema,
          this.rightColumnName);
97        this.rightColumnIndex = rightSchemaNames.indexOf(rightColumnName);
98        List<Type> leftSchemaTypes = new ArrayList<>(leftSchema.getFieldTypes());
99        List<Type> rightSchemaTypes = new ArrayList<>(rightSchema.getFieldTypes());
100       if
          (!leftSchemaTypes.get(this.leftColumnIndex).getClass().equals(rightSchemaTypes.ge
          t(
101           this.rightColumnIndex).getClass())) {
102         throw new QueryPlanException("Mismatched types of columns " + leftColumnName
            + " and "
103             + rightColumnName + ".");
104       }
105       leftSchemaNames.addAll(rightSchemaNames);
106       leftSchemaTypes.addAll(rightSchemaTypes);
107       return new Schema(leftSchemaNames, leftSchemaTypes);
108     }
109
110     public String str() {
111       return "type: " + this.joinType +
112               "\nleftColumn: " + this.leftColumnName +
113               "\nrightColumn: " + this.rightColumnName;
114     }
115
116     @Override
117     public String toString() {
118       String r = this.str();
119       if (this.leftSource != null) {
120         r += "\n" + ("(left)\n" + this.leftSource.toString()).replaceAll("(?m)^",
          "\t");
121       }
122       if (this.rightSource != null) {
123         if (this.leftSource != null) {
124           r += "\n";
125         }
```

```
        r += "\n" + ("(right)\n" + this.rightSource.toString()).replaceAll("(?m)^",
        "\t");
      }
      return r;
    }



    public byte[] getPageHeader(String tableName, Page p) throws DatabaseException {
      return this.transaction.readPageHeader(tableName, p);
    }

    public int getNumEntriesPerPage(String tableName) throws DatabaseException {
      return this.transaction.getNumEntriesPerPage(tableName);
    }

    public int getEntrySize(String tableName) throws DatabaseException {
      return this.transaction.getEntrySize(tableName);
    }

    public int getHeaderSize(String tableName) throws DatabaseException {
      return this.transaction.getPageHeaderSize(tableName);
    }

    public String getLeftColumnName() {
      return this.leftColumnName;
    }

    public String getRightColumnName() {
      return this.rightColumnName;
    }

    public Database.Transaction getTransaction() {
      return this.transaction;
    }

    public int getLeftColumnIndex() {
      return this.leftColumnIndex;
    }

    public int getRightColumnIndex() {
      return this.rightColumnIndex;
    }

    public Record getRecord(String tableName, RecordId rid) throws DatabaseException {
      return this.transaction.getRecord(tableName, rid);
    }

    public RecordIterator getRecordIterator(String tableName) throws
    DatabaseException {
      return this.transaction.getRecordIterator(tableName);
    }

    public BacktrackingIterator<Page> getPageIterator(String tableName) throws
    DatabaseException {
      return this.transaction.getPageIterator(tableName);
    }

    public BacktrackingIterator<Record> getBlockIterator(String tableName, Page[]
    block) throws DatabaseException {
      return this.transaction.getBlockIterator(tableName, block);
    }

    public BacktrackingIterator<Record> getBlockIterator(String tableName,
    BacktrackingIterator<Page> block) throws DatabaseException {
      return this.transaction.getBlockIterator(tableName, block);
```

```
188     }
189
190     public BacktrackingIterator<Record> getBlockIterator(String tableName,
        Iterator<Page> block, int maxPages) throws DatabaseException {
191       return this.transaction.getBlockIterator(tableName, block, maxPages);
192     }
193
194
195     public String createTempTable(Schema schema) throws DatabaseException {
196       return this.transaction.createTempTable(schema);
197     }
198
199     public void createTempTable(Schema schema, String tempTableName) throws
        DatabaseException {
200       this.transaction.createTempTable(schema, tempTableName);
201     }
202
203     public RecordId addRecord(String tableName, List<DataBox> values) throws
        DatabaseException {
204       return this.transaction.addRecord(tableName, values);
205     }
206
207     public JoinType getJoinType() {
208       return this.joinType;
209     }
210
211     /**
212      * All iterators for subclasses of JoinOperator should subclass from
213      * JoinIterator; JoinIterator handles creating temporary tables out of the left
        and right
214      * input operators.
215      */
216     protected abstract class JoinIterator implements Iterator<Record> {
217       private String leftTableName;
218       private String rightTableName;
219
220       public JoinIterator() throws QueryPlanException, DatabaseException {
221         if (JoinOperator.this.getLeftSource().isSequentialScan()) {
222           this.leftTableName = ((SequentialScanOperator)
            JoinOperator.this.getLeftSource()).getTableName();
223         } else {
224           this.leftTableName =
            JoinOperator.this.createTempTable(JoinOperator.this.getLeftSource().getOutput
            Schema());
225           Iterator<Record> leftIter = JoinOperator.this.getLeftSource().iterator();
226           while (leftIter.hasNext()) {
227             JoinOperator.this.addRecord(this.leftTableName,
              leftIter.next().getValues());
228           }
229         }
230         if (JoinOperator.this.getRightSource().isSequentialScan()) {
231           this.rightTableName = ((SequentialScanOperator)
            JoinOperator.this.getRightSource()).getTableName();
232         } else {
233           this.rightTableName =
            JoinOperator.this.createTempTable(JoinOperator.this.getRightSource().getOutpu
            tSchema());
234           Iterator<Record> rightIter = JoinOperator.this.getRightSource().iterator();
235           while (rightIter.hasNext()) {
236             JoinOperator.this.addRecord(this.rightTableName,
              rightIter.next().getValues());
237           }
238         }
239       }
240
241       protected String getLeftTableName() {
242         return this.leftTableName;
```

```
243         }
244
245         protected String getRightTableName() {
246           return this.rightTableName;
247         }
248       }
249     }
250
```