QUERY OPERATOR

```java
package query;

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

import database.DatabaseException;
import table.Record;
import table.Schema;

public abstract class QueryOperator {
  private QueryOperator source;
  private QueryOperator destination;
  private Schema operatorSchema;
  protected int cost;

  public enum OperatorType {
    JOIN,
    PROJECT,
    SELECT,
    GROUPBY,
    SEQSCAN,
    INDEXSCAN
  }

  private OperatorType type;

  public QueryOperator(OperatorType type) {
    this.type = type;
    this.source = null;
    this.operatorSchema = null;
    this.destination = null;
  }

  protected QueryOperator(OperatorType type, QueryOperator source) throws
  QueryPlanException {
    this.source = source;
    this.type = type;
    this.operatorSchema = this.computeSchema();
    this.destination = null;
  }

  public OperatorType getType() {
    return this.type;
  }

  public boolean isJoin() {
    return this.type.equals(OperatorType.JOIN);
  }

  public boolean isSelect() {
    return this.type.equals(OperatorType.SELECT);
  }

  public boolean isProject() {
    return this.type.equals(OperatorType.PROJECT);
  }

  public boolean isGroupBy() {
    return this.type.equals(OperatorType.GROUPBY);
  }

  public boolean isSequentialScan() {
    return this.type.equals(OperatorType.SEQSCAN);
  }

  public boolean isIndexScan() {
    return this.type.equals(OperatorType.INDEXSCAN);
    }

    public QueryOperator getSource() throws QueryPlanException {
      return this.source;
    }

    public QueryOperator getDestination() throws QueryPlanException {
      return this.destination;
    }

    public void setSource(QueryOperator source) throws QueryPlanException {
      this.source = source;
      this.operatorSchema = this.computeSchema();
    }

    public void setDestination(QueryOperator destination) throws QueryPlanException {
      this.destination = destination;
    }

    public Schema getOutputSchema() {
      return this.operatorSchema;
    }

    protected void setOutputSchema(Schema schema) {
      this.operatorSchema = schema;
    }

    protected abstract Schema computeSchema() throws QueryPlanException;

    public Iterator<Record> execute() throws QueryPlanException, DatabaseException {
      return iterator();
    }

    public Iterator<Record> execute(Object... arguments) throws QueryPlanException, DatabaseException {
      return null;
    }

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

    /**
     * Utility method that checks to see if a column is found in a schema using dot notation.
     *
     * @param fromSchema the schema to search in
     * @param specified the column name to search for
     * @return
     */
    public boolean checkColumnNameEquality(String fromSchema, String specified) {
      if (fromSchema.equals(specified)) {
        return true;
      }
      if (!specified.contains(".")) {
        String schemaColName = fromSchema;
        if (fromSchema.contains(".")) {
          String[] splits = fromSchema.split("\\.");
          schemaColName = splits[1];
        }

        return schemaColName.equals(specified);
      }
      return false;
    }

    /**
     * Utility method to determine whether or not a specified column name is valid
```

```java
	 with a given schema.
	 *
	 * @param schema
	 * @param columnName
	 * @return
	 * @throws QueryPlanException
	 */
	public String checkSchemaForColumn(Schema schema, String columnName) throws
	QueryPlanException {
	  List<String> schemaColumnNames = schema.getFieldNames();
	  boolean found = false;
	  String foundName = null;
	  for (String sourceColumnName : schemaColumnNames) {
	    if (this.checkColumnNameEquality(sourceColumnName, columnName)) {
	      if (found) {
	        throw new QueryPlanException("Column " + columnName + " specified twice
	        without disambiguation.");
	      }
	      found = true;
	      foundName = sourceColumnName;
	    }
	  }
	  if (!found) {
	    throw new QueryPlanException("No column " + columnName + " found.");
	  }
	  return foundName;
	}

	public String str() {
	  return "type: " + this.getType();
	}

	public String toString() {
	  String r = this.str();
	  if (this.source != null) {
	    r += "\n" + this.source.toString().replaceAll("(?m)^", "\t");
	  }
	  return r;
	}

}
```