CS3223 Database Systems Implementation
AY21/22 Semester 2
Project Report
Team 37

Prof Tan Kian-Lee

Team Members	Student No.	Email
Kelvin Wong	A0201706U	e0415515@u.nus.edu
Moon Geonsik	A0210908L	e0484312@u.nus.edu
Wincent Tjoi	A0201480W	e0412905@u.nus.edu

Github link: https://github.com/Wincenttjoi/CS3223-DBMS-Project

Bonuses

Bonus Feature 1: Range Search for Mergejoin, IndexJoin and IndexSelect

Note:

Implement range search for MergeJoin, IndexJoin and IndexSelect

- MergeJoin: All non-equi operators allowed except “!=”
- IndexJoin and IndexSelect:
  - If indextype is hash, only use index select scan when operator is "="
  - If indextype is btree and operator is "!=" or "<>", use normal product scan

File Changes	**Changes**
Planner Related Changes	
SimpleDBEngine/src/simpledb/opt/ **TablePlanner**.java	- updated **makeIndexJoin()** to allow non-equality search if the index supports range search. If it does, also ensure that it is not “!=” (or “<>”).

	- updated **makeMergeJoin()** to allow non-equality search and rearrange parameters in **MergeJoinPlan** based on lhs vs rhs. Also, flip operator if “>” and “>=” to “<” and “<=” respectively (explained in **MergeJoinScan**) - updated **makeIndexSelect()** to allow non-equality search if the index supports range search. If it does, also ensure that it is not “!=” (or “<>”).
SimpleDBEngine/src/simpledb/opt/ **MergeJoinScan**.java	- Renamed the default next() method to nextEquals() - Created a method named nextLesser() that handles inequality terms e.g. lhs < rhs or lhs <= rhs. It makes use of the fact that if both scans s1 and s2 are sorted, then if lhs1 < rhs1 is false, then lhs2 < rhs1 is also false. Thus, the position of s2 is saved when the inequality term holds true for a lhs’s first time. Subsequent lhs comparisons will start from the saved position of s2. - Created a method named next() that returns nextEquals or nextLesser depending on the operator
Index Related Changes	
SimpleDBEngine/src/simpledb/index/ **IndexInfo**.java	- added **supportRangeSearch(String opr)** method to return true if it is supported by the **idxtype**
SimpleDBEngine/src/simpledb/index/ **Index**.java SimpleDBEngine/src/simpledb/index/btree/ **BTPage**.java SimpleDBEngine/src/simpledb/index/btree/ **BTreeDir**.java	- overload **beforeFirst()**, **findSlotBefore()**, **findChildBlock()**, **search()** method to allow search with specific operator

SimpleDBEngine/src/simpledb/index/hash/**HashIndex**.java	
SimpleDBEngine/src/simpledb/index/btree/**BTreeIndex**.java	- fix **searchCost()** method of **BTreeIndex** to return 1 when numblocks == 0. The original code would have resulted in arithmetic error with an empty table.
SimpleDBEngine/src/simpledb/index/btree/**BTreeLeaf**.java	- modify **next()** and **tryOverflow()** method of **BTreeLeaf** to check when key matches via **OprComparator** instead of "="
SimpleDBEngine/src/simpledb/index/planner/**IndexSelectPlan**.java SimpleDBEngine/src/simpledb/index/query/**IndexSelectScan**.java	- added operator parameter in constructors - modified **beforeFirst()** method of **IndexSelectScan** to utilize the operator in the index's **beforeFirst()**.
Query Related Changes	
SimpleDBEngine/src/simpledb/materialize/**OprComparator**.java	- created a static compare method to compare 2 **Constant** with a given operator.
SimpleDBEngine/src/simpledb/query/**Predicate**.java	- add **getOperatorFromConstantComparison(String fldname)** and **getOperatorFromFieldComparison(String fldname)**
SimpleDBEngine/src/simpledb/query/**Term**.java	- added **getOperator()** method

Bonus Feature 2: Implementing a better query optimizer

Note:

Bonus 2a

The planner now considers all 4 possible implementations of join and chooses the one with lowest blocks accessed instead of simply choosing in the order of index join, hash join, merge join as suggested in Heuristic 7 of the textbook (default implementation of SimpleDB).

Bonus 2b

By default, SimpleDB's Heuristic Query Planner after creating index select plan and other join algorithms, it will create a Select Plan again on top of the same terms in the predicate.

For example, with majorid as a btree index, the query "select majorid, sid from student where majorid > 10 and sid = 4" will produce a plan of Index Select Plan on majorid > 10, and then Select Plan on majorid > 10 and sid = 4.

After our implementation to fix this, the plan used are "Index Select Plan on majorid > 10, Select Plan on sid = 4", where repetitions are avoided.

See: Feature 2: Before and after changes

Bonus 2c

After implementing bonus 2b, a query with multiple indexes in a single query would return the wrong result.

For example, if we have enroll (studentid) and student (sid) hash indexes, the query of “select studentid, sid from student, enroll where studentid = 1 and sid = 1” would return the wrong result, where sid attribute does not get filtered at all.

We further implement this to ensure that SimpleDB is working well as before.

File Changes	Changes
Planner Related Changes	
SimpleDBEngine/src/simpledb/opt/ **TablePlanner**.java	- added class variables **selectTermToRemove** and **joinTermsSelectedToRemove**, which store the terms to be excluded when returning plans for index select and corresponding join algorithms. - **makeSelectPlan** will return a plan with all applicable select terms, without any select term used by **makeIndexSelect**. This is done while maintaining the selected term in the **TablePlanner**’s predicate so that the **TablePlanner** can be reused.

	- **makeJoinPlan** will return a plan with all applicable join and select terms, without any **joinTerm** used by the join algorithm while maintaining the **joinTerm** in the **TablePlanner**'s predicate. This is done while maintaining the **joinTerm** in the **TablePlanner**'s predicate so that the **TablePlanner** can be reused. - modified **makeJoinPlan** routine to create all join plans, then sorting based on blocks accessed and choosing the plan with least estimated blocks accessed.
Query Related Changes	
SimpleDBEngine/src/simpledb/query/ **Term**.java	- added **matches(Term)** method to check if this term matches another term, regardless of direction. E.g. (a > 3) matches (3 < a) and (c >=d) matches (d>= c)
SimpleDBEngine/src/simpledb/query/ **Predicate**.java	- added new method **removeTerm(Term)** to remove all terms that match the parameter

Bonus Feature 3: Float data type which supports group by AVG

File Changes	Changes
Planner Related Changes	
SimpleDBEngine/src/simpledb/materialize/**GroupByPlan**.java	- refactored **GroupByPlan** constructor to add a float field to the schema for average aggregation function
SimpleDBEngine/src/simpledb/record/**Schema**.java	- created **addFloatField()** to support float data type field in schema
SimpleDBEngine/src/simpledb/jdbc/…/**ResultSet**.java (etc.) SimpleDBEngine/src/simpledb/…/**Plan**.java (etc.) SimpleDBEngine/src/simpledb/…/**Scan**.java (etc.)	- refactored to support float data type e.g. implement **getFloat()**, **setFloat()**, etc. - applies to any class that implements **getInt()**/**setString()** and **getString()**/**setString()**
SimpleDBEngine/src/simpledb/query/**Constant**.java	- refactored to hold float value and support comparison and hashing of float values
SimpleDBEngine/src/simpledb/materialize/**AvgFn**.java	- refactored to hold float value and return **Constant** initialised with float value

Bonus Feature 4: Joining an empty table with no tuples to another table

Note:

By default, SimpleDB will throw an error if a table with no tuples is joined with another table.

We have handled this gracefully that we will return an empty result to the user, since conceptually such a query can exist and a user wants to know that there is an empty result.

See: Feature 4: Before and after changes

File Changes	**Changes**
Planner Related Changes	
SimpleDBEngine/src/simpledb/materialize/**MaterializedPlan**.java	- in **open()**, we keep track of boolean **isTupleExist**, only when **src.next()** is true, **isTupleExist** is toggled to true. At the end of **open()**, **isTupleExist** will be checked, and if true, **setTupleExist** to be true in dest temporary table.
SimpleDBEngine/src/simpledb/multibuffer/**MultibufferProductScan**.java	- **next()** will have an additional guard clause to check if **lhsscan** have a tuple. If no tuple, return false.

SimpleDBEngine/src/simpledb/record/**TableScan**.java	- new class field of **isTupleExist** (boolean) for tracking purposes - upon construction of object, set **isTupleExist** to be false - when **next()** is called and returns true, set **isTupleExist** to true - added setter and getter for **isTupleExist**

Labs

Lab 2: Support for hash index and B+-tree index

File Changes	Changes
Index and Metadata	
SimpleDBEngine/src/simpledb/metadata/ **IndexInfo**.java	- stored **idxtype** as private field - new public setter method **setIdxType** to set private index - **open()** and **blocksAccessed()** to call hashindex / btreeindex depending on **idxtype** field of **IndexInfo**
SimpleDBEngine/src/simpledb/metadata/ **IndexMgr**.java	- **schema** to add another string field of “indextype” for **tblmgr** - **createIndex** takes in new parameter **idxtype**, which the **tableScan** ts will **setString** on “indextype” - **getIndexInfo** have **getString(“indextype”)** which will be used to **setIdxType** for **IndexInfo**
SimpleDBEngine/src/simpledb/metadata/ **MetadataMgr**.java	- createIndex have new parameter String **idxtype**, and **idxmgr.createIndex** will pass the **idxtype**
SimpleDBEngine/src/simpledb/metadata/ **MetadataMgrTest**.java	- **mdm.createIndex** to support hardcoded “hash”/”btree” as new parameters
SimpleDBEngine/src/simpledb/parse/ **CreateIndexData**.java	- supported new private field **idxtype** and updated constructor - getter method for **indexType()**

Parser Related Changes	
SimpleDBEngine/src/simpledb/parse/**Lexer**.java	- new private field of Collection<String> **idxType** - **initIdxType()** which add keywords of “hash” and “btree”, initialized during **Lexer** object creation - added “using” inside **initKeywords()** - implemented **matchIdxType** and **eatIdxType**, similar to other match and eat methods
SimpleDBEngine/src/simpledb/parse/**Parser**.java	- **createIndex()** will **eatKeyword**(“using”) - **createIndexData** takes in new parameter **idxtype**, which is retrieved by **lex.eatIdxType()**
Planner Related Changes	
SimpleDBEngine/src/simpledb/index/planner/**IndexUpdatePlanner**.java	- **executeCreateIndex** to **call mdm.createIndex** with a new parameter **indexType**
SimpleDBEngine/src/simpledb/plan/**BasicUpdatePlanner**.java	- **mdm.createIndex** to take in new parameter **data.indexType()**
SimpleDBEngine/src/simpledb/server/**SimpleDB**.java	- used **HeuristicQueryPlanner** and **IndexUpdatePlanner** instead of basic planners
Planner Related Changes	

CreateStudentDB.java	- added create index statements on student / enroll with the keyword “using hash/btree”

See also: Lab 2: Sequence Diagram for IdxMgr and TableScan

Lab 3: Support for order by clause and sorting

File Changes	Changes
Parser Related Changes	
SimpleDBEngine/src/simpledb/parse/**Lexer**.java	- added keywords “order” and “by” to the keywords list (in routine **initKeywords()**) - created a list of sort type (“asc,” “desc”) (in routine **initSortType()**) - added a new routine **eatSortType()** to take in the sort type and return a boolean value (“asc”=>true; “desc”=>false;) - added a new routine **matchSortType()** to check if the current token is a legal sort type (“asc,” “desc”)
SimpleDBEngine/src/simpledb/parse/**Parser**.java	- modified routine **query()** to detect the “order by” keyword and call **sortList()**. Throw a **BadSyntaxException**, if invalid “order by” syntax is used - added a new routine **sortList()** (called when the “order” keyword is detected) to parse the “order by” clause and to initialize and return a **LinkedHashMap<String, Boolean>**containing <key, value> pairs of the field to be sorted and whether it should be sorted in an ascending or descending order (order is set to ascending by default)
Planner Related Changes	

SimpleDBEngine/src/simpledb/opt/ **HeuristicQueryPlanner**.java	- modified the routine **createPlan()** to initialize and return a **SortPlan** object, using the **ProjectPlan** object, if the query contains an “order by” clause. Otherwise, return a **ProjectPlan** object
Sorting Related Changes	
SimpleDBEngine/src/simpledb/materialize/ **SortPlan**.java	- modified the constructor of **SortPlan**object that iterates over **sortfields** to assign a true Boolean value to each field as a key to create a **Map<String, Boolean>** object and initialize a **RecordComparator** object with the Map object - created a new constructor for **SortPlan**object that takes in a **Map<String, Boolean>** object, instead of a **List<String>** object, and initialize a **RecordComparator** object with the Map object
SimpleDBEngine/src/simpledb/materialize/ **RecordComparator**.java	- modified the constructor of **RecordComparator** object to take in a **Map<String, Boolean>** object and initialize the Map object as **sortMap**. Also, extract the keyset of the Map object as a **List<String>** and initialize the List object as **fields** - modified the routine **compare()** to check if the current field that the routine is iterating over should be sorted in an “ascending” or “descending” order using the **sortMap** and return the integer result. (if the order is descending, change the sign of the integer result and return)

Lab 4: Supporting for nested-loops join, sort-merge join and index-based join

File Changes	Changes
Parser Related Changes	
SimpleDBEngine/src/simpledb/parse/**Parser**.java	- added ability to parse keywords ‘join’ and ‘on’ in **tableList()** and extract the table name and predicate
Planner Related Changes	
SimpleDBEngine/src/simpledb/opt/**TablePlanner**.java	- added **makeMergeJoin** routine to create **MergeJoinPlan** - added **makeNestedJoin** routine to create **NestedJoinPlan** - modified **makeJoinPlan** routine to allow either **indexJoinPlan**, **mergeJoinPlan** or **nestedJoinPlan** to be created based on heuristic 7 of the textbook
SimpleDBEngine/src/simpledb/materialize/**NestedJoinScan**.java SimpleDBEngine/src/simpledb/materialize/**NestedJoinPlan**.java	- Created nestedJoinPlan and NestedJoinScan based on ProductPlan, and checking the predicate for each pair of records.
Sorting Related Changes	

SimpleDBEngine/src/simpledb/materialize/**SortPlan**.java	- modified condition in **open()** to end generation of sorted runs when size <= 1 instead of 2. - added **blocksAccessedWithSortCost()** method to estimate number of block accesses for sorting and materialization with the formula taught in syllabus
SimpleDBEngine/src/simpledb/materialize/**MergeJoinPlan**.java	- updated **blocksAccessed()** method to include the sorting cost of both plans using **SortPlan.blocksAccessedWithSortCost()**, assuming best case scenario
Query Related Changes	
SimpleDBEngine/src/simpledb/query/**Constant**.java	- **equals()** returns false if obj being compared is null
SimpleDBEngine/src/simpledb/query/**Predicate**.java	- added get method for list of terms
SimpleDBEngine/src/simpledb/query/**Term**.java	- added get method for LHS and RHS expressions

Lab 5: Support for partition-based (or hash) join, and aggregates (SUM, COUNT, AGV, MIN, MAX) with/without group by clause (using sort-based implementation of group by operator)

File Changes	**Changes**
Parser Related Changes	
SimpleDBEngine/src/simpledb/parse/ **Parser**.java	- created **aggField()** to extract information from the aggregate field - refactored **selectList()** to parse the select fields and properly extract information about the aggregation function from the aggregate fields - created **parseAggFn()** to initialize and return the correct **AggregationFunction** - created **groupList()** to extract information from the group by fields - refactored **query()** to parse aggregate fields and group by fields
SimpleDBEngine/src/simpledb/parse/ **Lexer**.java	- added keywords “group” and “hashjoin” to the keywords list (in routine **initKeywords()**) - created a list of aggregation type (“sum”, “count”, “avg”, “min,” “max”) (in routine **initAggType()**) - added a new routine **eatAggType()** to take in the aggregation type and return a string value - added a new routine **matchAggType()** to check if the current token is a legal aggregation type

SimpleDBEngine/src/simpledb/parse/**QueryData**.java	- refactored **QueryData** to initialize a list of aggregation functions and a list of group by fields - refactored **toString()** to print out information about aggregate fields and group by fields
Group By Related Changes	
SimpleDBEngine/src/simpledb/opt/**HeuristicQueryPlanner**.java	- refactored **createPlan()** to aggregate results on group fields, if either aggregate fields or group by fields are included in the query
SimpleDBEngine/src/simpledb/materialize/**AvgFn**.java	- created **AvgFn** class to compute the average of values in the same group - **processFirst()** initializes the sum as the value of the first record and initializes the num as 1 - **processNext()** accumulates the sum by the value of the current record and increments the num - **value()** returns a **Constant** object initialized with sum divided by num
SimpleDBEngine/src/simpledb/materialize/**MinFn**.java	- created **MinFn** class to compute the minimum of values in the same group - **processFirst()** initializes the minimum as the value of the first record - **processNext()** updates the minimum to be the value of the current record, if the value is less than the current minimum - **value()** returns a **Constant** object initialized with the current minimum
SimpleDBEngine/src/simpledb/materialize/**SumFn**.java	- created **SumFn** class to compute the sum of values in the same group - **processFirst()** initializes the sum as the value of the first record - **processNext()** accumulates the sum by the value of the current record - **value()** returns a **Constant** object initialized with the current sum

Hash Join Related Changes	
SimpleDBEngine/src/simpledb/opt/**TablePlanner**.java	- refactored **makeJoinPlan()** to initialize a **HashJoinPlan** and compare the number of accessed blocks with other joins - created **makeHashJoin()** to execute hash join, if the operator of the predicate is equality and enough buffer size is available (B > sqrt(M)). Otherwise, execute product join
SimpleDBEngine/src/simpledb/materialize/**HashComparator**.java	- created **HashComparator** class to check if the two scans are hashed to the same bucket, based on the values of the specified fields
SimpleDBEngine/src/simpledb/materialize/**HashJoinPlan**.java	- created **HashJoinPlan** class to initialize a plan that joins two tables using hashjoin method - **open()** initializes a hashjoin scan and hashes the record of each table into k number of partitions (temporary tables) using **splitIntoPartitions()** - **splitIntoPartitions()** initializes k number of partitions (temporary tables) and scans the given table to hash each record into the corresponding partitions
SimpleDBEngine/src/simpledb/materialize/**HashJoinScan**.java	- created **HashJoinScan** class to initialize a scan that probes two matching partitions and finds the matching records using a different hash function - **next()** traverses through all sets of matching partitions and returns true when a set of matching records that are hashed to the same hash table are found

Lab 6: Support for DISTINCT and displaying the query plan

File Changes	Changes
Parser Related Changes	
SimpleDBEngine/src/simpledb/parse/**Lexer**.java	- added keywords “distinct” to the keywords list (in routine **initKeywords()**)
SimpleDBEngine/src/simpledb/parse/**Parser**.java	- added ability to parse keywords ‘distinct’ in **query()** after eating keyword ‘distinct’
Planner Related Changes	
SimpleDBEngine/src/simpledb/opt/**HeuristicQueryPlanner**.java	- added another condition to check if **QueryData isDistinct** field is true, then calls sort plan and distinctplan.
SimpleDBEngine/src/simpledb/materialize/**DistinctPlan**.java	- **DistinctPlan** implements **Plan**, which calls **DistinctScan(Scan, fields)** when **open()** is called.
SimpleDBEngine/src/simpledb/materialize/**DistinctScan**.java	- contains private fields of **Scan** s, **List<String>** fields, and **List<Constant>** of prev and curr (to keep track of current and previous record values) - calls internal method **nextDistinct()** when **next()** is called

	- **nextDistinct()** returns a boolean, and calls **s.next()** recursively if the next record is a duplicate, returns false when no more record. Each field of the projected query has their **Constant** value compared, to know whether the next record is a duplicate.
Query Related Changes	
SimpleDBEngine/src/simpledb/parse/**QueryData**.java	- holds a new boolean field **isDistinct** - **isDistinct** field supported during **QueryData** object initialization - getter method for **isDistinct** - **toString** method to include **isDistinct**, if any
Displaying Query Related Changes	
SimpleDBEngine/src/simpledb/index/planner/**IndexJoinPlan**.java SimpleDBEngine/src/simpledb/index/planner/**IndexSelectPlan**.java SimpleDBEngine/src/simpledb/materialize/**DistinctPlan**.java SimpleDBEngine/src/simpledb/materialize/**GroupByPlan**.java SimpleDBEngine/src/simpledb/materialize/**MaterializePlan**.java	- all classes that implement **Plan** will have to implement **printPlan()** method, which tells users that the plan is being used. Specific details could also be shown when necessary. - every **open()** method will call **printPlan()** method, after opening the scan further in.

SimpleDBEngine/src/simpledb/materialize/**MergeJoinPlan**.java SimpleDBEngine/src/simpledb/materialize/**SortPlan**.java SimpleDBEngine/src/simpledb/multibuffer/**MultibufferProductPlan**.java SimpleDBEngine/src/simpledb/plan/**OptimizedProductPlan**.java SimpleDBEngine/src/simpledb/plan/**ProductPlan**.java SimpleDBEngine/src/simpledb/plan/**ProjectPlan**.java SimpleDBEngine/src/simpledb/plan/**SelectPlan**.java SimpleDBEngine/src/simpledb/plan/**TablePlan**.java	
SimpleDBEngine/src/simpledb/plan/**Plan**.java	- interface **Plan** to have **printPlan()** method, which displays to the user that the plan is used.

See also: Lab 6: Sequence Diagram for Distinct, Lab 6: Classes implementing Plan

Lab 7: Integrate all features into SimpleDB+

Note:

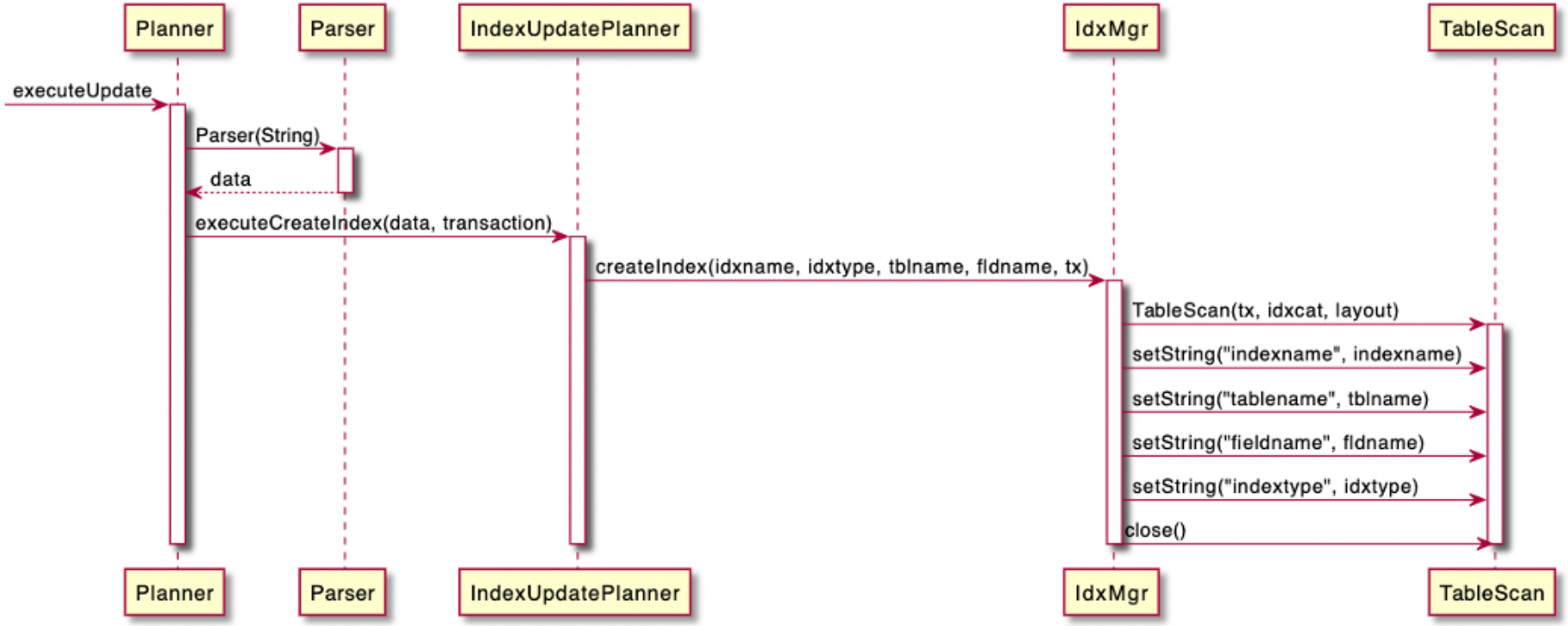
1. We have been continuously integrating each lab, so there is not much effort needed for integration here.
2. Add range search for nestedJoin.
3. Heuristic Planner will be used by default, Basic Planner no longer used.

File Changes	**Changes**
Parser Related Changes	
SimpleDBEngine/src/simpledb/parse/**Lexer**.java SimpleDBEngine/src/simpledb/parse/**Parser**.java	- added “indexjoin”, “mergejoin”, “nestedjoin” and “hashjoin” as keywords - added information on selected join algorithm to data if the above keywords are found before ‘select’ keyword
Planner Related Changes	

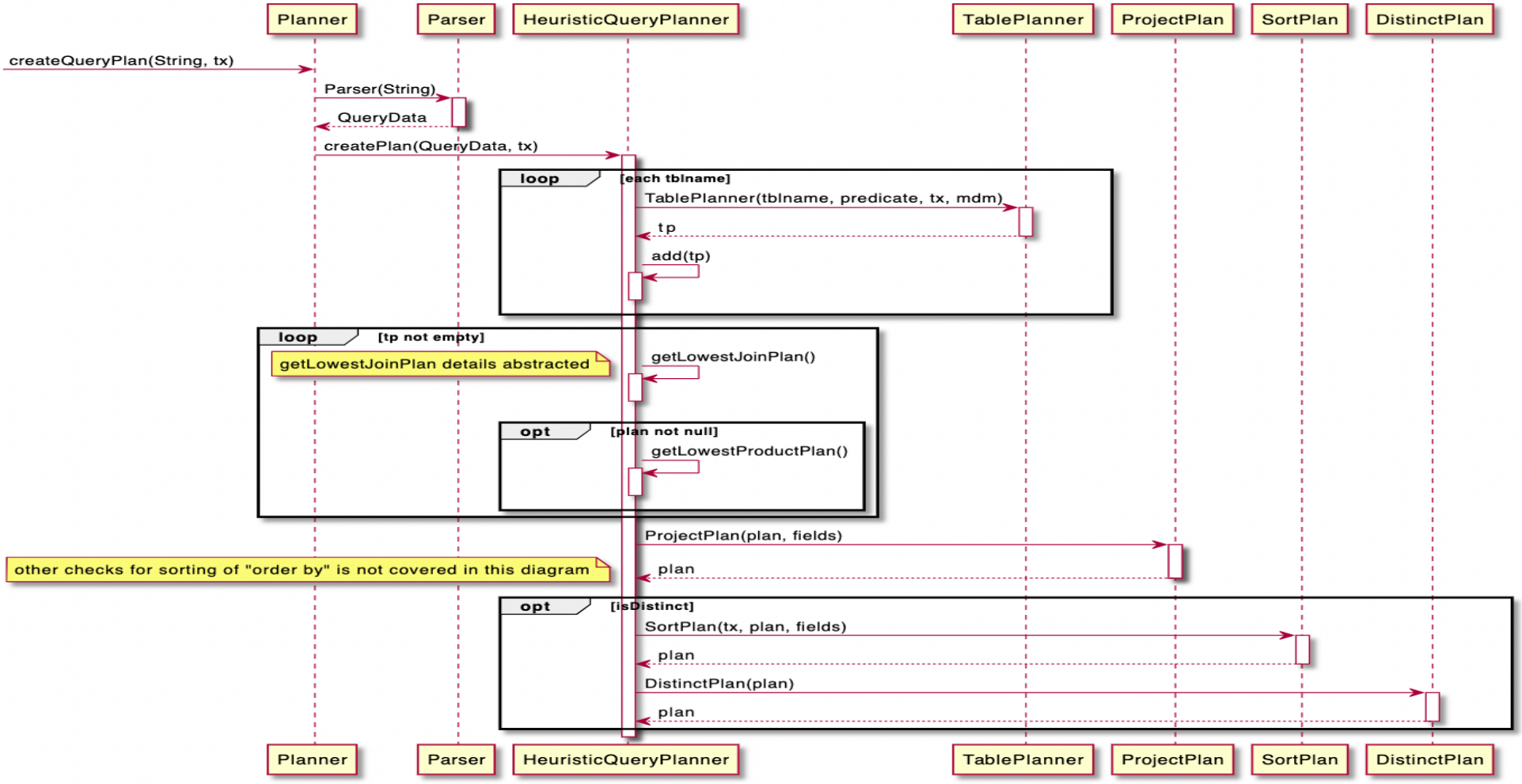
SimpleDBEngine/src/simpledb/opt/**TablePlanner**.java SimpleDBEngine/src/simpledb/opt/**HeuristicQueryPlanner**.java	- updated to allow a specific join algorithm to be chosen for showcase in future

Appendix

Lab 2: Sequence Diagram for IdxMgr and TableScan

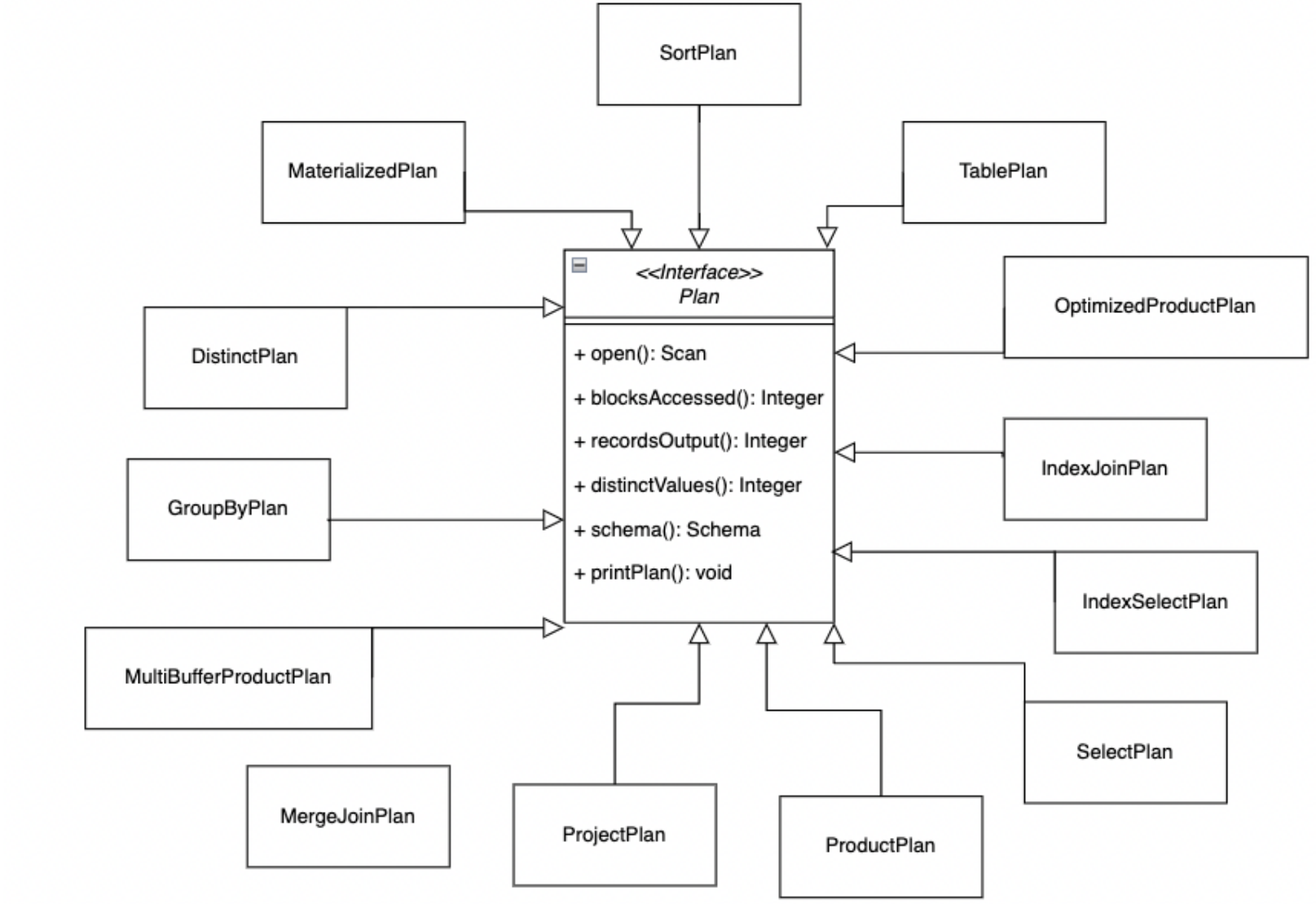


Lab 6: Sequence Diagram for Distinct



Note: HashJoinPlan, NestedJoinPlan and other plans are excluded for brevity.

Lab 6: Classes implementing Plan



Feature 2: Before and after changes

```
111          Test.doTest(stmt, "select sname, sid, majorid from student where majorid >= 20 and sid > 4 and sname != 'bob'");
112  // TEST9
113  //          sname    sid majorid
114  //          -------------------------
115  //                   kim      6      20
116  //                   art      7      30
117  //                   pat      8      20
```

Console | Problems | Debug Shell | Search

```
<terminated> Lab1Test (1) [Java Application] /Users/wincenttjoi/Library/Java/JavaVirtualMachines/openjdk-17.0.1/Contents/Home/bin/java  (15 Mar 2022, 10:56:22 pm – 10:56:23 pm)
TEST9
Table Plan used on table student
Index Select Plan used for table student of field majorid
Select Plan used on predicate majorid>=20 and sid>4 and sname!=bob
Project Plan used on fields of sname, sid, majorid
      sname    sid majorid
-------------------------
        pat      8      20
        kim      6      20
        art      7      30
The total time taken for query is: 747375ns
transaction 10 committed
```

IndexSelect: Before

```
TEST9: select sname, sid, majorid from student where majorid >= 20 and sid > 4 and sname != 'bob'
Table Plan used on table student
Index Select Plan used for table student of field majorid>=20, index accessed: majorid_idx, index type: btree
Select Plan used on predicate sid>4 and sname!=bob
Project Plan used on fields of sname, sid, majorid
The total time taken for query is: 400583ns
      sname    sid majorid
-------------------------
        pat      8      20
        kim      6      20
        art      7      30
transaction 10 committed
=========================================
```

IndexSelect: After

```
TEST4: indexjoin select sname, sid, studentid, grade from student, enroll where studentid = sid
>>Indexjoin blocks accessed = 13
Table Plan used on table enroll
Table Plan used on table student
Index join Plan used on sid=studentid
Select Plan used on predicate studentid=sid
Project Plan used on fields of sname, sid, studentid, grade
The total time taken for query is: 3ms
      sname    sid studentid grade
-----------------------------------
        joe      1         1     A
        joe      1         1     C
        amy      2         2    B+
        sue      4         4     B
        sue      4         4     A
        kim      6         6     A
transaction 11 committed
----------------------------------------
```

IndexJoin: Before

```
TEST4: indexjoin select sname, sid, studentid, grade from student, enroll where studentid = sid
>>Indexjoin blocks accessed = 13
Table Plan used on table enroll
Table Plan used on table student
Index join Plan used on sid=studentid
Project Plan used on fields of sname, sid, studentid, grade
The total time taken for query is: 2ms
      sname    sid studentid grade
-----------------------------------
        joe      1         1     A
        joe      1         1     C
        amy      2         2    B+
        sue      4         4     B
        sue      4         4     A
        kim      6         6     A
```

IndexJoin: After

Feature 4: Before and after changes

```
TEST17: select sname from student, dept where sname = 'jonah'
Table Plan used on table student
Select Plan used on predicate sname=jonah
Materialized Plan created with fields [sid, sname, majorid, gradyear]
Table Plan used on table dept
Multibuffer Product Plan used between simpledb.materialize.MaterializePlan@6d311334 and simpledb.plan.TablePlan@682a0b20
Project Plan used on fields of sname
The total time taken for query is: 2ms
      sname
-----------
transaction 18 rolled back
SQL Exception: java.lang.IllegalArgumentException: newPosition < 0: (-22 < 0)
```

Joining with an empty table : Before

```
TEST17: select sname from student, dept where sname = 'jonah'
Table Plan used on table student
Select Plan used on predicate sname=jonah
Materialized Plan created with fields [sid, sname, majorid, gradyear]
Table Plan used on table dept
Multibuffer Product Plan used between simpledb.materialize.MaterializePlan@682a0b20 and simpledb.plan.TablePlan@3d075dc0
Project Plan used on fields of sname
The total time taken for query is: 2ms
      sname
-----------
transaction 18 committed
========================================
```

Joining with an empty table : After