[430-555] Homework 3 : Join Operator

Student Name: LE VAN DUC **Student ID: 2016-27885**

1. System configuration:

CPU: Intel® Core™ i5-4460 CPU @ 3.20GHz × 4

OS: Ubuntu 14.04 LTS 64 bit

RAM: 8 GB

HDD: 1.1 TB

2. Join Operator Design

2.1. Catalog

*** Tables (Relations) catalog:**

Orders: #Rows = 1,500,000.

Fields	Type (size in bytes)	Primary key	Position
o_orderkey	INT (4)	**true**	1
o_orderstatus	CHAR (1)	false	2
o_totalprice	INT (4)	false	3
o_comment	VARCHAR (79)	false	4

Lineitem: #Rows = 6,001,215

Fields	Type (size in bytes)	Foreign key	Position
l_orderkey	INT (4)	ON Orders (o_orderkey)	1
l_quantity	INT (4)		2
l_returnflag	CHAR (1)		3

*** Column-store catalog:**

Columns	Dictionary Type	#DistinctValues = Dictionary Size	Dictionary Sorted ?
o_orderkey	INT	1,500,000	**true**
o_orderstatus	STRING	3	false
o_totalprice	INT	347,282	**true**
o_comment	STRING	1,482,071	false
l_orderkey	INT	1,500,000	**true**
l_quantity	INT	50	**true**
l_returnflag	STRING	3	false

*** Indexes catalog:**

Columns	Type	Dictionary Size	#Words (inverted index)
o_comment	Inverted Index	1,482,071	10,696,973

2.2. Operator Input/Output

*** Input:**

- Input is 2 relations (Orders and Lineitem) and the columns to join. In this case we will join 2 columns Orders.o_orderkey with Lineitem.l_orderkey at their equal values. Because each column is encoded by dictionary encoding, so we cannot compare their values directly but through a mapping between 2 columns' encoded value.

Orders relation:

rowId (implicit)	o_orderkey's encoded value	o_orderkey's dictionary
1	0 (0000)	138890
2	0 (0000)	138891
3	4 (0010)	138892
...	...	...

Lineitem relation:

rowId (implicit)	l_orderkey's encoded value	l_orderkey's dictionary
1	1 (0001)	138890
2	1 (0001)	138891
3	0 (0000)	138892
...	...	...

Mapping between o_orderkey's encoded values and l_orderkey's encoded values:

o_orderkey's encoded values	l_orderkey's encoded values
0	1
1	3
4	0
...	...

*** Output:**

- The output relation will be the concatenation of 2 input relations with rows are the matching by equal values between 2 join columns.

o_orderkey (= l_orderkey)	o_status	o_totalprice	o_comment	l_quantity	l_returnflag
138890	"O"	82880	"..."	70	"N"
...	...	...	...	...	...

- Then the final result will be the projection of the output relation to some columns. In this Homework we will project output relation to all Orders' columns (o_orderkey, o_status, o_totalprice, o_comment).

o_orderkey	**o_status**	**o_totalprice**	**o_comment**
138890	"O"	82880	"..."
...	...	...	...

2.3. Intermediate Result Representation

- The JOIN operator will be the result after some Selection operator (if existed) on joining table to reduce the number of rows to join.

Joining Table
row1
row2
row3
row4
row5

=> (Selected by some conditions)

Joining Table
row1
row2
row3

- The **Intermediate result** of JOIN operator will be **a relation with concatenated columns** of joining relation by join condition. If we have more than 2 join relations then the result of 2 joining relation will be continued to join with other relation by other join condition.

o_orderkey (=l_orderkey)	**l_orderkey (=o_orderkey)**	**o_status**	**o_totalprice**	**o_comment**	**l_quantity**	**l_returnflag**
key1	key1	status1	totalprice1	comment1	quantity1	returnflag1
key2	key2	status2	totalprice2	comment2	quantity2	returnflag2
...		...	...	...	...	...

3. Join Implementation:

3.1. Algorithm:

- There are some Join algorithms such as: Nested Loop Join, Nested Block Join, Hash Join, Merge Join. In this case of Column-Store DB, I will use **Hash Join algorithm**.

- Algorithm description:

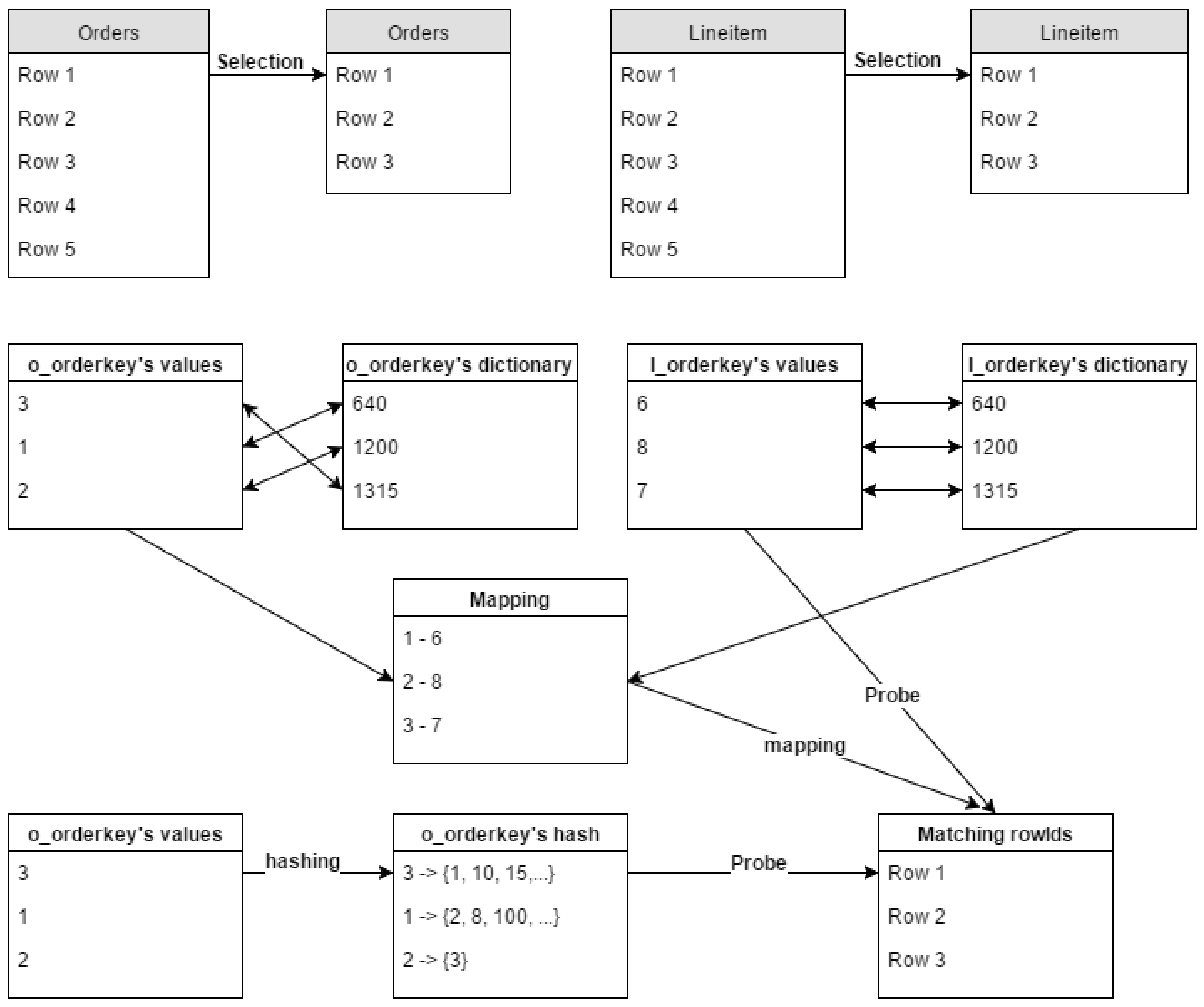
+) **Step 1**: Do the **selection** (where condition) on each table to **reduce the number of rows** of each table to join. Output: the list of rowIds of each table to join.

+) **Step 2**: In case of Column-Store DB, because we use Dictionary encoding then we need to create a **mapping** between the values of 2 columns to join through their Dictionary. In this case we will create a mapping of the values of Orders.o_orderkey and Lineitem.l_orderkey through their dictionary because 2 columns' Dictionaries have equal values.

+) **Step 3** (**hash phase**): Between the list of rowIds of 2 joining table, choose the **smaller to HASH**. Here I will use **C++ map data structure** to create a **hashmap** between the **values of joining column** with **one or many corresponding rowIds** (because each value of hash column can appear in many rows). For example, if the rowIds list of Orders.o_orderkey is smaller then we will create a hashmap of each value of column o_orderkey with 1 or many rows that this value appears.

+) **Step 4** (**probe phase**): For each value of **other joining column** (values corresponding with the selection result in Step 1), we will **match (probe) with the hashed value** of 1st column by using the mapping in Step 2. If this value is in hashed The **list of pair of matching rowIds between 2 tables** will be brought to the JOIN result. The query result will be projected based on the pair of rowIds list of JOIN result.

- Algorithm steps figure:



3.2. Coding implement: - update 2016/11/02

- Step 0: Initialize join result matching row id pairs (App.cpp)

```
// initialize join result pairs of row ids
vector<tuple<int, int>>* join_result_pairs = new vector<tuple<int, int>>();
// initialize row ids result for selection
vector<bool>* l_rowIds = new vector<bool>();
for (size_t i = 0; i < l_orderkey->vecValueSize(); i++) {
        l_rowIds->push_back(false);
}
vector<bool>* o_rowIds = new vector<bool>();
for (size_t i = 0;i < o_orderkey->vecValueSize(); i++) {
        o_rowIds->push_back(false);
}
```

- Step 1: Selection (App.cpp)

```
// join example 2: orders.o_totalprice < 56789 AND l_quantity > 40
if (query.find("2") != string::npos) {
        Column<int>* o_totalprice = (Column<int>*) table-
>getColumnByName("o_totalprice");
        Column<int>* l_quantity = (Column<int>*) table2-
>getColumnByName("l_quantity");
        // execute where query
        int value = 56789;
        o_totalprice->selection(value, ColumnBase::ltOp, o_rowIds);
        value = 40;
        l_quantity->selection(value, ColumnBase::gtOp, l_rowIds);
}
// join example 3: orders.o_comment contains 'gift'
else if (query.find("3") != string::npos) {
        Column<string>* o_comment = (Column<string>*) table-
>getColumnByName("o_comment");
        // execute where query
        string value = "gift";
        o_comment->selection(value, ColumnBase::containOp, o_rowIds);
}
// join example 1: no where selection
else {
        // all rows of 2 joining tables are selected
        for (size_t i = 0; i < l_rowIds->size(); i++) {
                l_rowIds->at(i) = true;
        }
        for (size_t i = 0;i < o_rowIds->size(); i++) {
                o_rowIds->at(i) = true;
        }
}
```

- Step 2: mapping values of 2 joining columns (App.cpp)

```
// create mapping between vecValue of 2 join columns
map<size_t, size_t> mappingValueId;
for (size_t i = 0; i < l_orderkey->getDictionary()->size(); i++) {
        size_t valueId1 = i;
        int* dictValue1 = l_orderkey->getDictionary()->lookup(valueId1);
        if (dictValue1 != NULL) {
                // search dictionary value on 2nd column
                vector<size_t> result;
                o_orderkey->getDictionary()->search(*dictValue1,
ColumnBase::equalOp, result);
                // if not existed valueId2 then return -1
                size_t valueId2 = result[0];
                mappingValueId[valueId1] = valueId2;
        }
}
```

- Step 3: hashing (file: Column.h)

```
// Build hashmap of valueId based on selected row ids
void buildHashmap(map<size_t, vector<size_t>>& hashmap, vector<bool>*
vecRowId) {
        hashmap.clear();
        for (size_t rowId = 0; rowId < vecRowId->size(); rowId++) {
                // get valueId from bit packing if row id is selected
                // then build hashmap
                if (vecRowId->at(rowId)) {
                        size_t valueId = vecValueAt(rowId);
                        hashmap[valueId].push_back(rowId);
                }
        }
}
```

- Step 4: probe (App.cpp)

```
// probe (join) to find matching row ids
for (size_t rowId = 0; rowId < l_orderkey->vecValueSize(); rowId++) {
        // by pass if this rowId not included in the selection phase
        if (!l_rowIds->at(rowId)) continue;
        size_t valueId1 = l_orderkey->vecValueAt(rowId);
        // get valueId2 from mapping
        size_t valueId2 = mappingValueId[valueId1];
        // found on hashmap
        vector<size_t> rowIds = hashmap[valueId2];
        if (rowIds.size() > 0) {
                // make join result pairs
                for (size_t o_rowId : rowIds) {
                join_result_pairs->push_back(make_tuple(l_rowId, o_rowId));
                }

        }
}
```

4. Text type column storage layout design description

Column	**Type**	**Dictionary encoding**	**Inverted Index**
o_orderstatus	char (1)	true	false
o_comment	varchar (79)	true	**true**
l_returnflag	char (1)	true	false

- The dictionary of each text-type column will not be sorted because we do not need to binary search on it. We can use Inverted index to search more effective and faster.

- Only text-type column "**o_comment**" has Inverted Index because its values are long text (~ 79 characters) and has few repeating values => selection conditions on this column will be "contain" text search. Other columns are similar to “status” column, so they have very short and few distinct values => selection conditions will be "equal" search.

- Inverted Index on o_comment column's Dictionary:

+ Each comment Dictionary entry will be split by whitespace to **list of words**.

+ Each word then will be **normalized** to: **lowercase** and **reduced to word stem** to remove the effect of variance word forms (e.g.: plural, adjective...). I haven't applied stop-word removal yet but I will remove words which its length less than 4 (mostly stop words and no meaning words).

+ Then we will create **Inverted Index** from normalized word to the **Dictionary's position** of the Dictionary entry that contains the word. I haven't created an Inverted Index of word and its position in the text because it is not used in this Homework but the procedure is similar.

+ Inverted Index will be **sorted by word** to encourage the **binary search** on it.

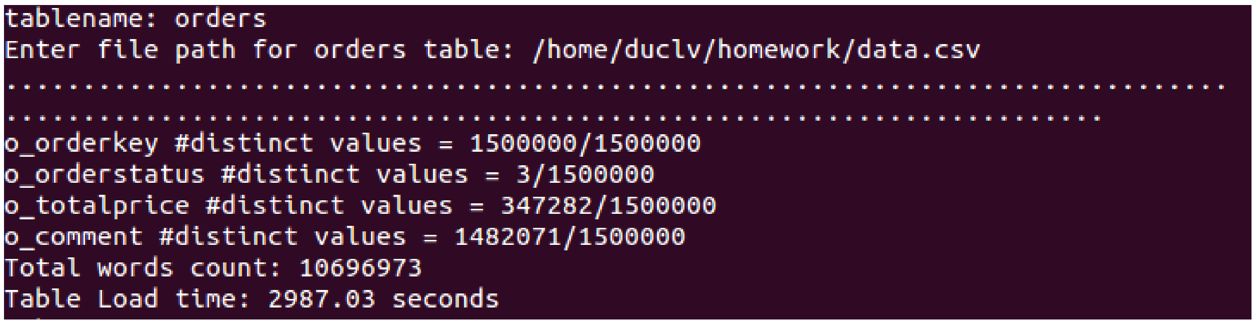
+ The search procedure is using **Binary search** to search on vector of Inverted Index and then returns the **list of text's locations** satisfying with the search value.

- Inverted Index coding implement (file Dictionary.cpp): Here I use a **Porter stemming** algorithm's implementation by Sean Massung (https://bitbucket.org/smassung/porter2_stemmer/wiki/Home). Inverted Index is stored as a C++ struct.

```
void Dictionary<string>::buildInvertedIndex() {
        // make an unordered_map of words from all items
        vector<string>* strItems = (vector<string>*) items;
        unordered_map<string, vector<size_t>> mapWordsLevel0;
        size_t wordCount = 0;
        for (size_t i = 0; i < strItems->size(); i++) {
                // split item into word by whitespace
                vector<string> words;
                istringstream iss(strItems->at(i));
                copy(istream_iterator<string>(iss),
istream_iterator<string>(), back_inserter(words));
                // add to map
                for (size_t j = 0; j < words.size(); j++) {
                        string word = words[j];
                        Porter2Stemmer::trim(word);    // normalize word
                        Porter2Stemmer::stem(word);    // stem by porter
algorithm
                        vector<size_t> locationLevel0 = mapWordsLevel0[word];
                        locationLevel0.push_back(i);
                        mapWordsLevel0[word] = locationLevel0;
                }
                wordCount += words.size();
        }
        cout << "Total words count: " << wordCount << endl;
        // create vector of inverted index level 0 from map
        for (const auto& m : mapWordsLevel0) {
                string word = m.first;
                vector<size_t> location = m.second;
                // inverted index level 0
                invertedIndex idxLevel0;
                idxLevel0.word = word;
                idxLevel0.location = location;
                vecIndexLevel0->push_back(idxLevel0);
        }
        // sort vector of inverted index
        std::sort(vecIndexLevel0->begin(), vecIndexLevel0->end());
}
```

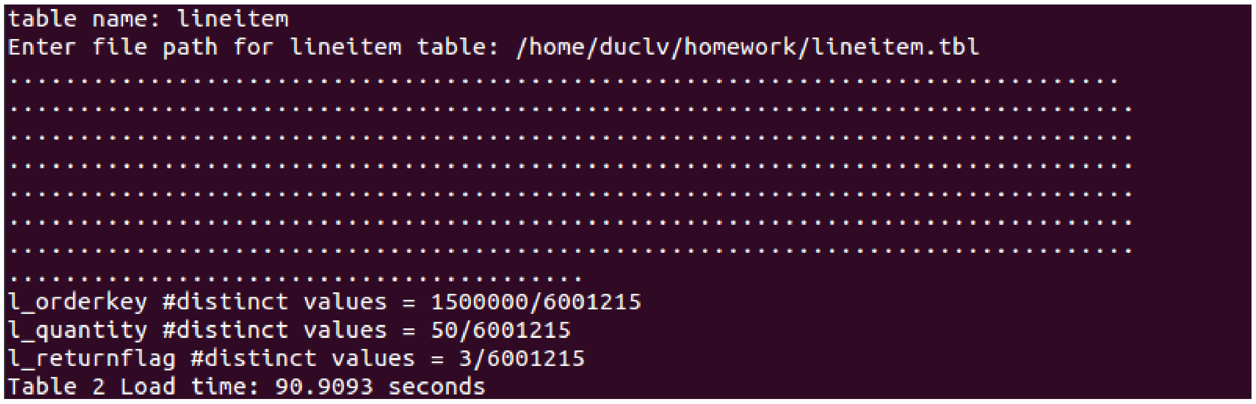
5. Experiment on 3 Join example queries:

Table Orders (1,500,000 rows) load time with Inverted Index created: ~ 2987 s



```
tablename: orders
Enter file path for orders table: /home/duclv/homework/data.csv
................................................................................
.......................................................................
o_orderkey #distinct values = 1500000/1500000
o_orderstatus #distinct values = 3/1500000
o_totalprice #distinct values = 347282/1500000
o_comment #distinct values = 1482071/1500000
Total words count: 10696973
Table Load time: 2987.03 seconds
```

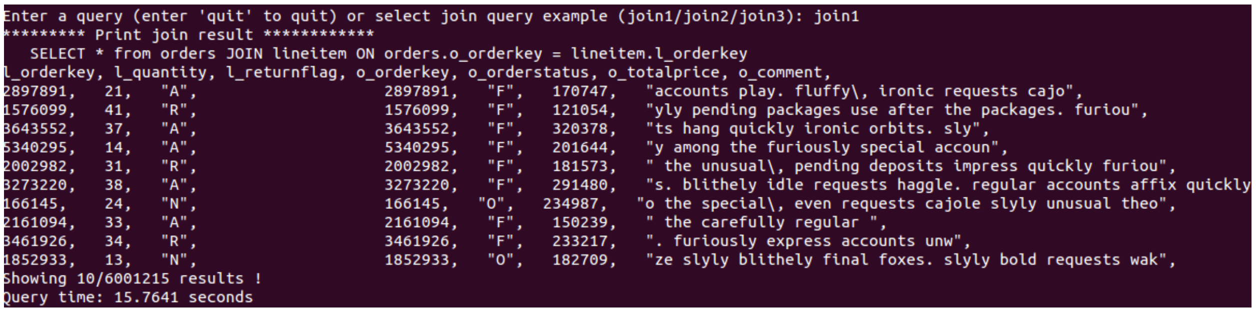
Table Lineitem (6,001,215 rows) load time without Inverted Index: ~ 91 s



```
table name: lineitem
Enter file path for lineitem table: /home/duclv/homework/lineitem.tbl
........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
....................................
l_orderkey #distinct values = 1500000/6001215
l_quantity #distinct values = 50/6001215
l_returnflag #distinct values = 3/6001215
Table 2 Load time: 90.9093 seconds
```

Comment: The time it takes to create an Inverted index is much greater than the time to create Dictionary encoding for each column.

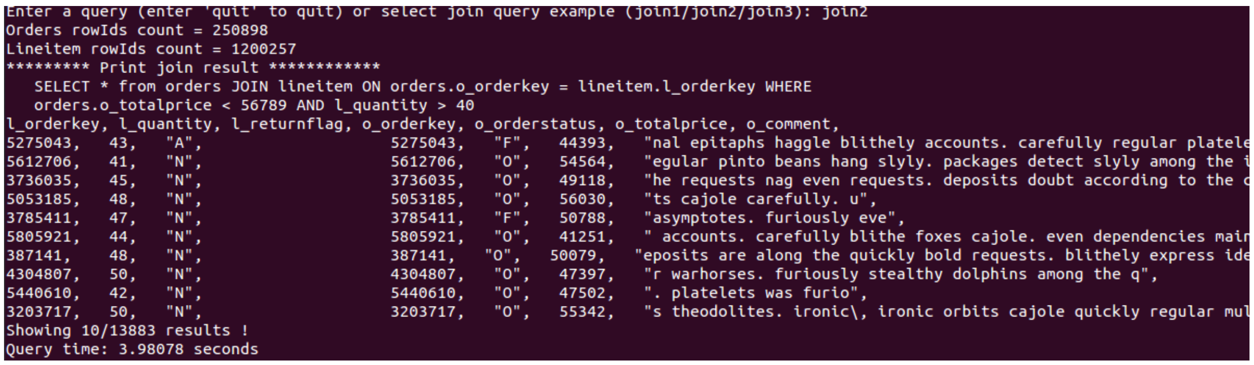
a) Query 1: SELECT * from orders JOIN lineitem ON orders.o_orderkey = lineitem.l_orderkey => **6,001,215 rows**, average ~ **15.74s**



```
Enter a query (enter 'quit' to quit) or select join query example (join1/join2/join3): join1
********* Print join result ************
   SELECT * from orders JOIN lineitem ON orders.o_orderkey = lineitem.l_orderkey
l_orderkey, l_quantity, l_returnflag, o_orderkey, o_orderstatus, o_totalprice, o_comment,
2897891,   21,   "A",                    2897891,   "F",   170747,   "accounts play. fluffy\, ironic requests cajo",
1576099,   41,   "R",                    1576099,   "F",   121054,   "yly pending packages use after the packages. furiou",
3643552,   37,   "A",                    3643552,   "F",   320378,   "ts hang quickly ironic orbits. sly",
5340295,   14,   "A",                    5340295,   "F",   201644,   "y among the furiously special accoun",
2002982,   31,   "R",                    2002982,   "F",   181573,   " the unusual\, pending deposits impress quickly furiou",
3273220,   38,   "A",                    3273220,   "F",   291480,   "s. blithely idle requests haggle. regular accounts affix quickly
166145,    24,   "N",                    166145,   "O",   234987,   "o the special\, even requests cajole slyly unusual theo",
2161094,   33,   "A",                    2161094,   "F",   150239,   " the carefully regular ",
3461926,   34,   "R",                    3461926,   "F",   233217,   ". furiously express accounts unw",
1852933,   13,   "N",                    1852933,   "O",   182709,   "ze slyly blithely final foxes. slyly bold requests wak",
Showing 10/6001215 results !
Query time: 15.7641 seconds
```

b) Query 2: SELECT * from orders JOIN lineitem ON orders.o_orderkey = lineitem.l_orderkey WHERE orders.o_totalprice < 56789 AND l_quantity > 40

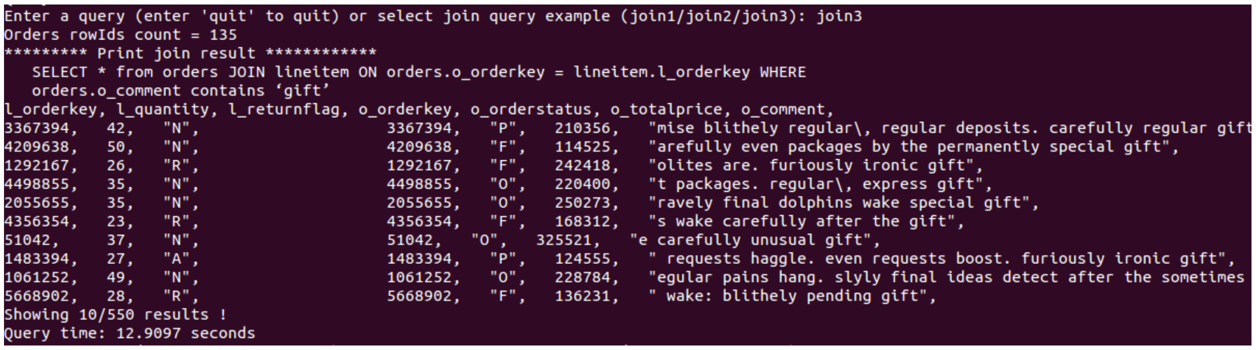
=> **13,883 rows**, average ~ **3.86s**



```
Enter a query (enter 'quit' to quit) or select join query example (join1/join2/join3): join2
Orders rowIds count = 250898
Lineitem rowIds count = 1200257
********* Print join result ************
   SELECT * from orders JOIN lineitem ON orders.o_orderkey = lineitem.l_orderkey WHERE
   orders.o_totalprice < 56789 AND l_quantity > 40
l_orderkey, l_quantity, l_returnflag, o_orderkey, o_orderstatus, o_totalprice, o_comment,
5275043,   43,   "A",                    5275043,   "F",   44393,   "nal epitaphs haggle blithely accounts. carefully regular platele
5612706,   41,   "N",                    5612706,   "O",   54564,   "egular pinto beans hang slyly. packages detect slyly among the i
3736035,   45,   "N",                    3736035,   "O",   49118,   "he requests nag even requests. deposits doubt according to the c
5053185,   48,   "N",                    5053185,   "O",   56030,   "ts cajole carefully. u",
3785411,   47,   "N",                    3785411,   "F",   50788,   "asymptotes. furiously eve",
5805921,   44,   "N",                    5805921,   "O",   41251,   " accounts. carefully blithe foxes cajole. even dependencies main
387141,    48,   "N",                    387141,   "O",   50079,   "eposits are along the quickly bold requests. blithely express ide
4304807,   50,   "N",                    4304807,   "O",   47397,   "r warhorses. furiously stealthy dolphins among the q",
5440610,   42,   "N",                    5440610,   "O",   47502,   ". platelets was furio",
3203717,   50,   "N",                    3203717,   "O",   55342,   "s theodolites. ironic\, ironic orbits cajole quickly regular mul
Showing 10/13883 results !
Query time: 3.98078 seconds
```

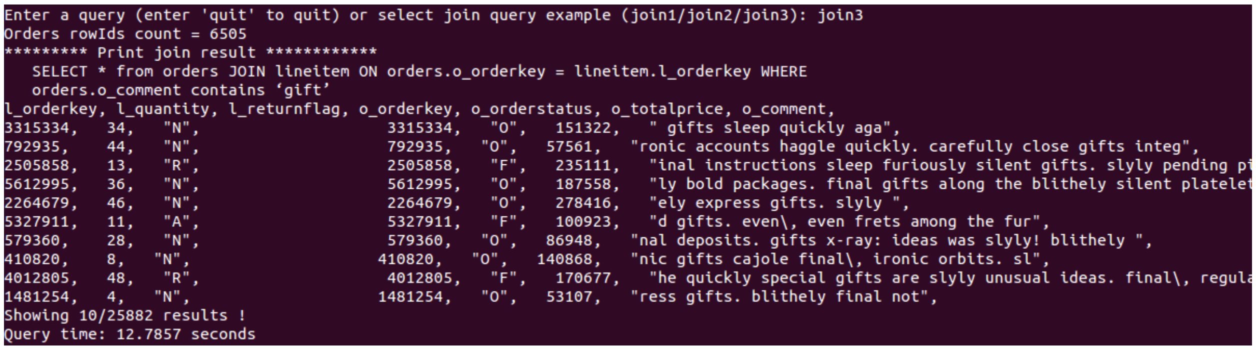
c) Query 3: SELECT * from orders JOIN lineitem ON orders.o_orderkey = lineitem.l_orderkey WHERE orders.o_comment contains ‘gift’

Case 1: not use Porter word stemming: search exact “gift” => **550 rows,** average **~ 12.91s**



```
Enter a query (enter 'quit' to quit) or select join query example (join1/join2/join3): join3
Orders rowIds count = 135
********* Print join result ************
   SELECT * from orders JOIN lineitem ON orders.o_orderkey = lineitem.l_orderkey WHERE
   orders.o_comment contains ‘gift’
l_orderkey, l_quantity, l_returnflag, o_orderkey, o_orderstatus, o_totalprice, o_comment,
3367394,   42,   "N",                    3367394,   "P",   210356,   "mise blithely regular\, regular deposits. carefully regular gift
4209638,   50,   "N",                    4209638,   "F",   114525,   "arefully even packages by the permanently special gift",
1292167,   26,   "R",                    1292167,   "F",   242418,   "olites are. furiously ironic gift",
4498855,   35,   "N",                    4498855,   "O",   220400,   "t packages. regular\, express gift",
2055655,   35,   "N",                    2055655,   "O",   250273,   "ravely final dolphins wake special gift",
4356354,   23,   "R",                    4356354,   "F",   168312,   "s wake carefully after the gift",
51042,     37,   "N",                    51042,   "O",   325521,   "e carefully unusual gift",
1483394,   27,   "A",                    1483394,   "P",   124555,   " requests haggle. even requests boost. furiously ironic gift",
1061252,   49,   "N",                    1061252,   "O",   228784,   "egular pains hang. slyly final ideas detect after the sometimes
5668902,   28,   "R",                    5668902,   "F",   136231,   " wake: blithely pending gift",
Showing 10/550 results !
Query time: 12.9097 seconds
```

Case 2: use Porter word stemming: search like {“gift”, “gifts”} => **25,882 rows**, avg ~ **12.78s**



```
Enter a query (enter 'quit' to quit) or select join query example (join1/join2/join3): join3
Orders rowIds count = 6505
********* Print join result ************
   SELECT * from orders JOIN lineitem ON orders.o_orderkey = lineitem.l_orderkey WHERE
   orders.o_comment contains ‘gift’
l_orderkey, l_quantity, l_returnflag, o_orderkey, o_orderstatus, o_totalprice, o_comment,
3315334,   34,   "N",                    3315334,   "O",   151322,   " gifts sleep quickly aga",
792935,    44,   "N",                    792935,   "O",   57561,   "ronic accounts haggle quickly. carefully close gifts integ",
2505858,   13,   "R",                    2505858,   "F",   235111,   "inal instructions sleep furiously silent gifts. slyly pending pi
5612995,   36,   "N",                    5612995,   "O",   187558,   "ly bold packages. final gifts along the blithely silent platelet
2264679,   46,   "N",                    2264679,   "O",   278416,   "ely express gifts. slyly ",
5327911,   11,   "A",                    5327911,   "F",   100923,   "d gifts. even\, even frets among the fur",
579360,    28,   "N",                    579360,   "O",   86948,   "nal deposits. gifts x-ray: ideas was slyly! blithely ",
410820,    8,   "N",                    410820,   "O",   140868,   "nic gifts cajole final\, ironic orbits. sl",
4012805,   48,   "R",                    4012805,   "F",   170677,   "he quickly special gifts are slyly unusual ideas. final\, regula
1481254,   4,   "N",                    1481254,   "O",   53107,   "ress gifts. blithely final not",
Showing 10/25882 results !
Query time: 12.7857 seconds
```