[430-555] Homework 2 : A Simple Column Store

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. C++ compiler configuration:

*) g++: version 4.8

*) CXXFLAGS = -O2 -g -Wall -fmessage-length=0 -std=c++11 -lstdc++ -lsqlparser -L/usr/local/lib/ -I/home/duclv/work/sql-parser/src/ -I/usr/local/boost_1_61_0

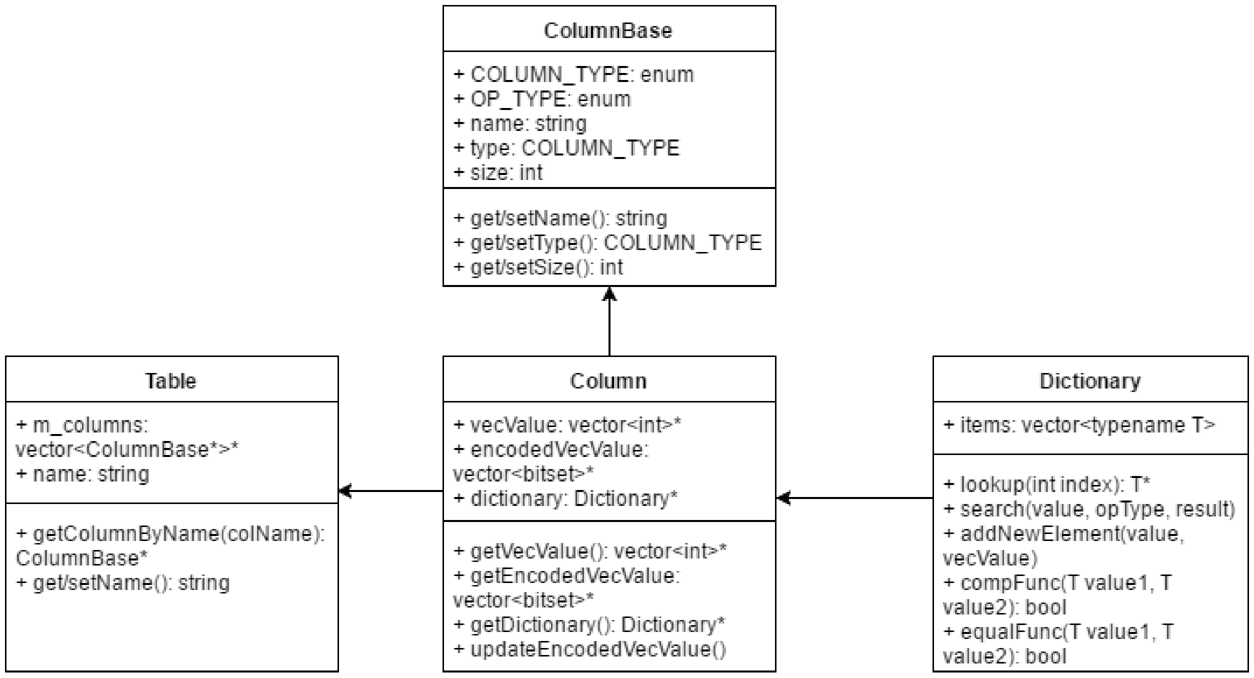
*) Use boost library at: http://www.boost.org/ => compile config: -I/usr/local/boost_1_61_0

*) Use open source SQL parser at: https://github.com/hyrise/sql-parser => compile config: -L/usr/local/lib/ -I/home/duclv/work/sql-parser/src/

*) Use open source bit packing PackedArray (https://github.com/gpakosz/PackedArray): already added header and source file into my source code.

3. Column Table description:

3.1. Architecture:


ColumnBase
+ COLUMN_TYPE: enum
+ OP_TYPE: enum
+ name: string
+ type: COLUMN_TYPE
+ size: int
+ get/setName(): string
+ get/setType(): COLUMN_TYPE
+ get/setSize(): int
Table
+ m_columns: vector<ColumnBase*>*
+ name: string
+ getColumnByName(colName): ColumnBase*
+ get/setName(): string
Column
+ vecValue: vector<int>*
+ encodedVecValue: vector<bitset>*
+ dictionary: Dictionary*
+ getVecValue(): vector<int>*
+ getEncodedVecValue: vector<bitset>*
+ getDictionary(): Dictionary*
+ updateEncodedVecValue()
Dictionary
+ items: vector<typename T>
+ lookup(int index): T*
+ search(value, opType, result)
+ addNewElement(value, vecValue)
+ compFunc(T value1, T value2): bool
+ equalFunc(T value1, T value2): bool

a) ColumnBase: base class for all columns.

+ Enums:

+ COLUMN_TYPE: column types: intType, charType, varcharType.

+ OP_TYPE: operator types: equalOp (=), neOp (<>), gtOp (>), geOp (>=), ltOp (<), leOp (<=), likeOp (like).

+ Fields:

+ string name: column name.

+ COLUMN_TYPE type: column type.

+ int size: column type size.

b) Dictionary: class to store and process dictionary encoding data

+ typename<T>: can keep any type (currently: int and string) which can support comparison operator.

+ Functions:

+ lookup(index): return an value from dictionary at position index.

+ search(value, opType, result): search value from dictionary based on opType (<, >, =,...), return result into a reference.

+ addNewElement(value, vecValue): add value into dictionary. This value can be existed on dictionary. After adding, update vecValue which is the vector value of column.

+ size(): get number of elements on dictionary.

+ compFunc(T value1, T value2): comparison function used to compare 2 values in typename T

+ equalFunc(T value1, T value2): function to check 2 values in typename T equal or not.

c) Column: class representing for a Column of table. Subclass of ColumnBase.

+ Members:

  + vecValue: a vector keeps values encoded by the dictionary of column.

  + packed: a dynamic array of bits used to pack/unpack vecValue. Use PackedArray open source.

  + dictionay: the dictionary data of column.

+ Functions:

  + getVecValue(): return the value vector of this column.

  + bitPackingVecValue(): packing value vector of this column into array of bit.

  + getDictionary(): return the dictionary vector of this column.

  + vecValueAt(index): return unpacked value of column’s vector value at index.

d) Table: class representing for a Table.

+ Members:

  + m_columns: a vector keeps all columns of the table.

  + name: table name

+ Functions:

  + getColumnByName(columnName): return the pointer to the column which has name equals to columnName.

3.2. Data structures and algorithms:

- With Dictionary: use **vector<T>** structure to keep encode dictionary data. In which, T is a **typename** which allow to store any type into dictionary (in case of Homework2, I will store integer and string). Moreover, to support search and insert on dictionary, I supply 2 functions to do the **comparison** (**compFunc(value1, value2)**) and **equalization** (**equalFunc(value1, value2)**).

- To keep **Dictionary sorted**, I use C++ function: **lower_bound()** to find appropriate position to insert new value into dictionary. I will keep sorting dictionary right on inserting new value. This function has the complexity **~ O(log2(N) + 1)** which **N is the size of the vector**.

- For **bit packing**: use an open source **PackedArray** to **pack/unpacking** values into/from an **array of bits**. Number of bit required = **#Values * #bits_required_to_present_max_values**.

- To **search** on **sorted dictionary** I also use function: **lower_bound()** to find **position** of satisfied value on dictionary then return all desired positions based on search condition.

- With column **o_comment** which has type of **varchar**, I will **not** keep sorted dictionary because most query with this column will use "**like**" similar operator so sorted dictionary will not work for this.

4. Column Table memory discussion:

Column	**Type**	**Size (bytes)**	**No. of Distinct Values (Size of dictionary)**
o_orderkey	integer	4	0
o_orderstatus	char	1	3
o_totalprice	integer	4	347,282
o_comment	varchar	79 (maximum)	1,475,124

- Table total records: 1,500,000 ~ **1.43M**.

- Total memory needed when using row-store: max ~ (4 + 1 + 4 + 79) * 1.43M ~ **125.84 MB**. Actually file size on disk ~ 105MB.

- With column-store, using dictionary encoding for columns which have many repeating values such as: **o_orderstatus, o_totalprice, o_comment**. (Not use with o_orderkey because it is primary key), we have:

+ **o_orderstatus**:

+ Uncompressed size: 1 * 1.43 M ~ 1.43 MB.

+ Compressed size:

+ Encoded value size: [log2(3)] ~ 2 bit.

+ Encoded column size: 2 * 1.43M ~ 2.86 Mbit ~ 0.36 MB

+ Dictionary size: 1 * 3 = 3 Bytes

⇨ Compression ratio: **0.36/1.43 ~ 25%**

+ **o_totalprice**:

+ Uncompressed size: 4 * 1.43 ~ 5.72 MB

+ Compressed size:

+ Encoded value size: [log2(347282)] ~ 18.4 ~ 19 bit

+ Encoded column size: 19 * 1.43M ~ 27.17 Mbit ~ 3.4 MB

+ Dictionary size: 4 * 347282 ~ 1.32 MB

⇨ Compression ratio**: 4.72/5.72 ~ 82.5%**

+ **o_comment**:

+ Uncompressed size: ~ 79 * 1.43 ~ 113 MB

+ Compressed size:

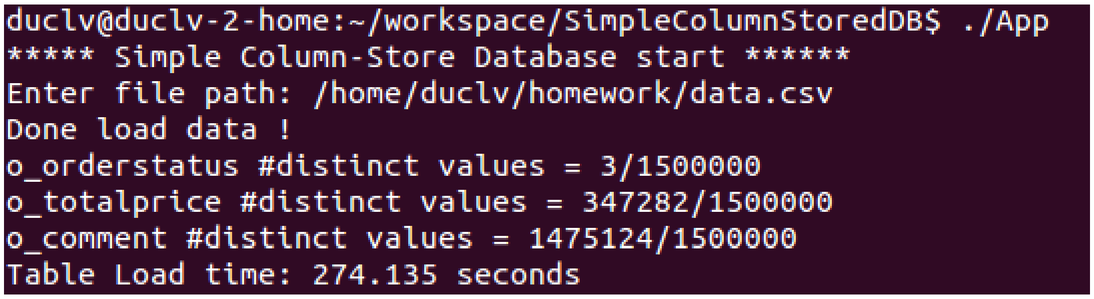
+ Encoded value size: [log2(1475124)] ~ 20.5 ~ 21 bit

+ Encoded column size: 21 * 1.43 ~ 30.03 Mbit ~ 3.75 MB

+ Dictionary size: ~ 79 * 1475124 ~ 111.15 MB

⇨ Compression ratio: **114.9/113 ~ 101% => column o_comment has no memory affection when using dictionary compression.**

5. Table Load (~ 274s):

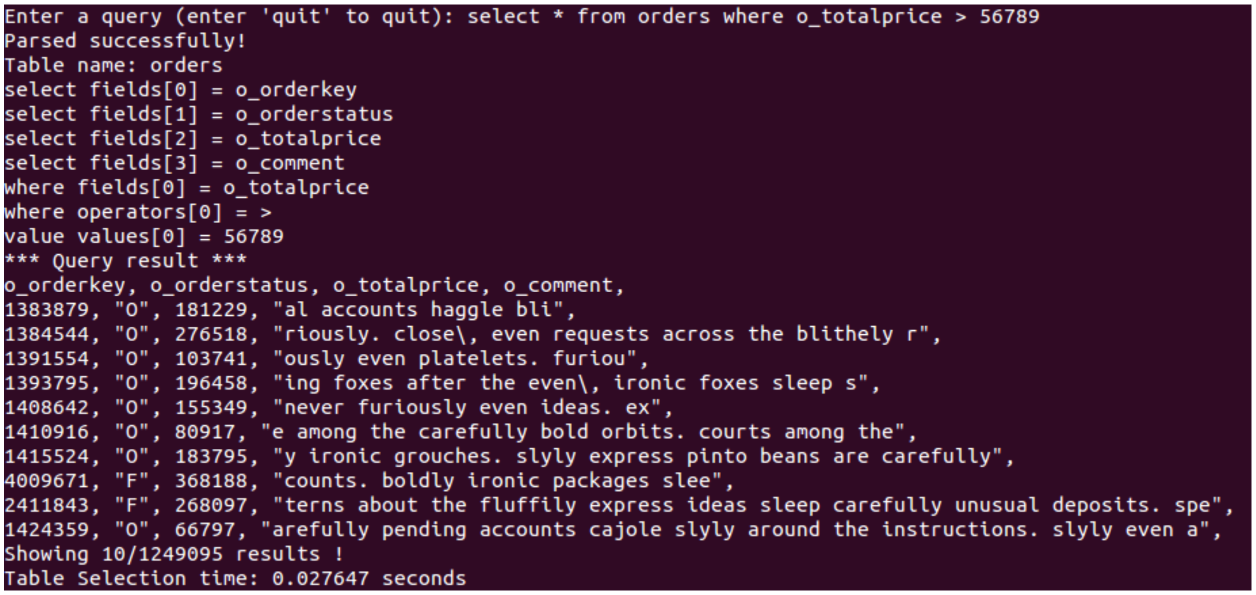


```
duclv@duclv-2-home:~/workspace/SimpleColumnStoredDB$ ./App
***** Simple Column-Store Database start ******
Enter file path: /home/duclv/homework/data.csv
Done load data !
o_orderstatus #distinct values = 3/1500000
o_totalprice #distinct values = 347282/1500000
o_comment #distinct values = 1475124/1500000
Table Load time: 274.135 seconds
```

6. Query execution time:

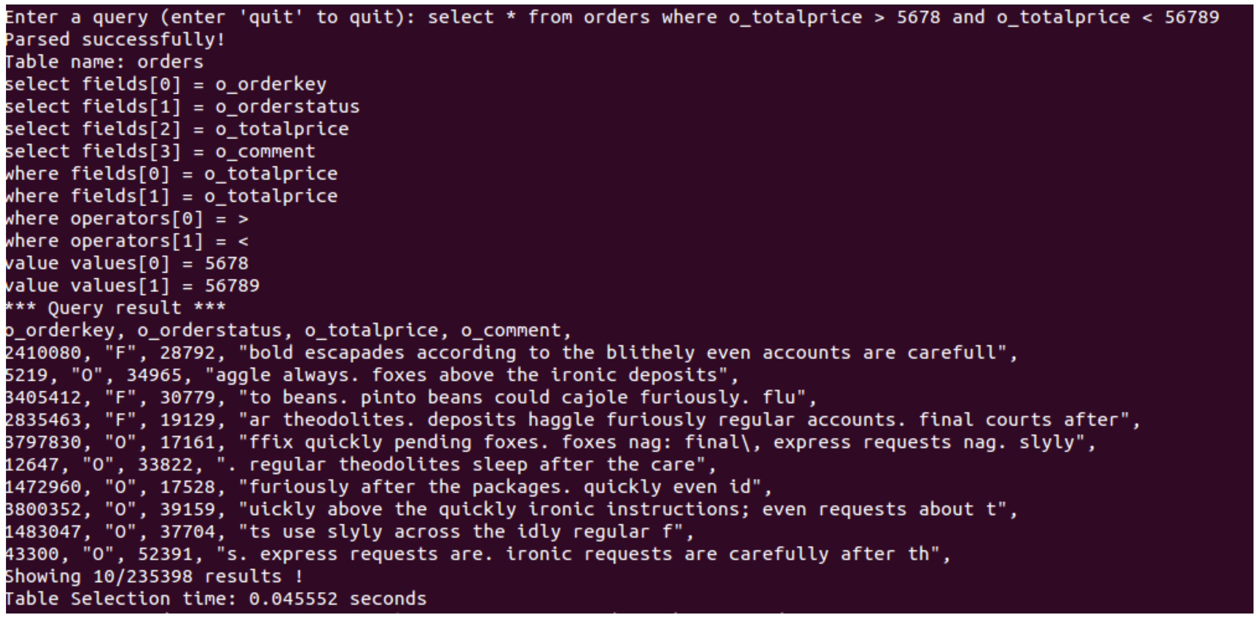
Support query by **operator: <, >, =** and query on **o_totalprice & o_orderstatus** field**.**

a) Query 1: select * from orders where o_totalprice > 56789 => ~ **0.028s**



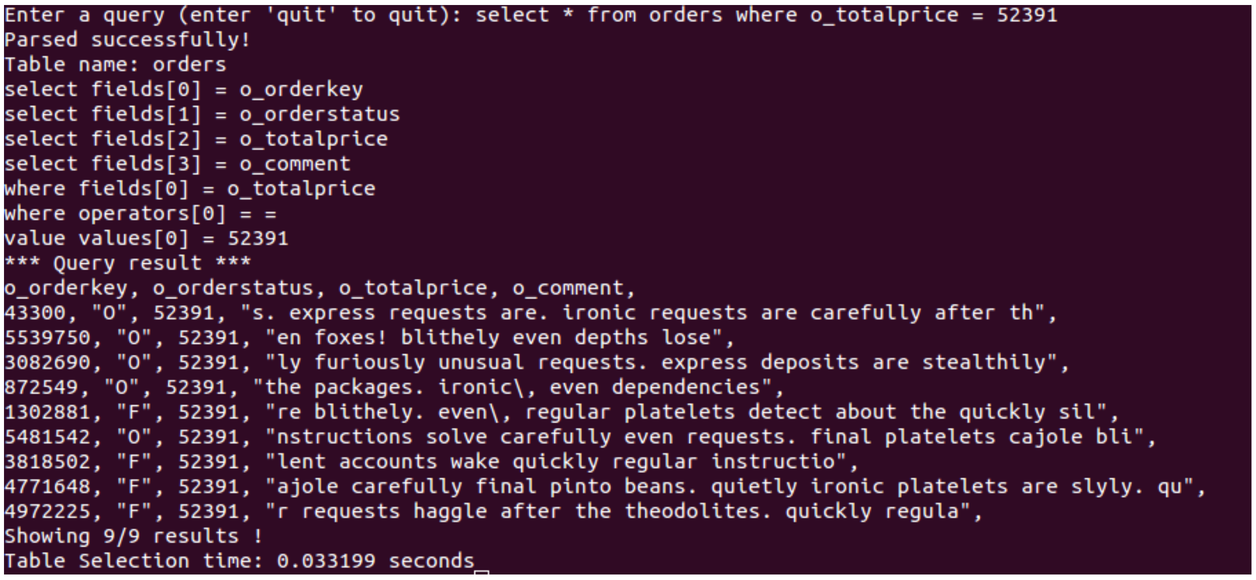
```
Enter a query (enter 'quit' to quit): select * from orders where o_totalprice > 56789
Parsed successfully!
Table name: orders
select fields[0] = o_orderkey
select fields[1] = o_orderstatus
select fields[2] = o_totalprice
select fields[3] = o_comment
where fields[0] = o_totalprice
where operators[0] = >
value values[0] = 56789
*** Query result ***
o_orderkey, o_orderstatus, o_totalprice, o_comment,
1383879, "O", 181229, "al accounts haggle bli",
1384544, "O", 276518, "riously. close\, even requests across the blithely r",
1391554, "O", 103741, "ously even platelets. furiou",
1393795, "O", 196458, "ing foxes after the even\, ironic foxes sleep s",
1408642, "O", 155349, "never furiously even ideas. ex",
1410916, "O", 80917, "e among the carefully bold orbits. courts among the",
1415524, "O", 183795, "y ironic grouches. slyly express pinto beans are carefully",
4009671, "F", 368188, "counts. boldly ironic packages slee",
2411843, "F", 268097, "terns about the fluffily express ideas sleep carefully unusual deposits. spe",
1424359, "O", 66797, "arefully pending accounts cajole slyly around the instructions. slyly even a",
Showing 10/1249095 results !
Table Selection time: 0.027647 seconds
```

b) Query 2: select * from orders where o_totalprice > 5678 and o_totalprice < 56789 => ~ **0.046s**



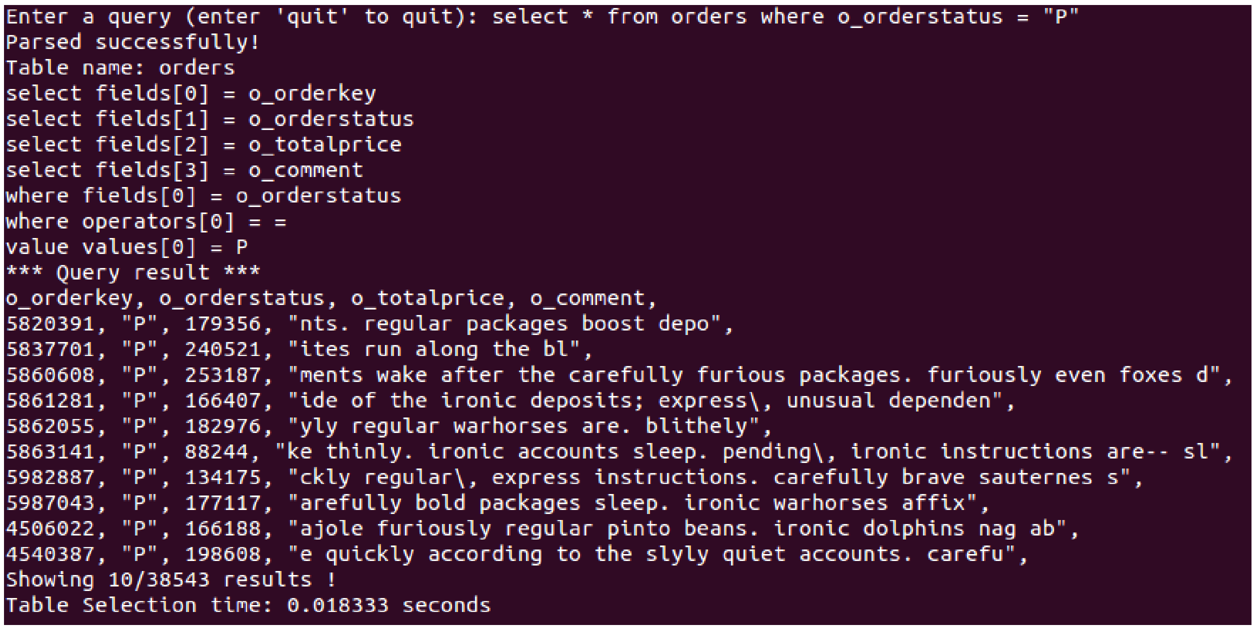
```
Enter a query (enter 'quit' to quit): select * from orders where o_totalprice > 5678 and o_totalprice < 56789
Parsed successfully!
Table name: orders
select fields[0] = o_orderkey
select fields[1] = o_orderstatus
select fields[2] = o_totalprice
select fields[3] = o_comment
where fields[0] = o_totalprice
where fields[1] = o_totalprice
where operators[0] = >
where operators[1] = <
value values[0] = 5678
value values[1] = 56789
*** Query result ***
o_orderkey, o_orderstatus, o_totalprice, o_comment,
2410080, "F", 28792, "bold escapades according to the blithely even accounts are carefull",
5219, "O", 34965, "aggle always. foxes above the ironic deposits",
3405412, "F", 30779, "to beans. pinto beans could cajole furiously. flu",
2835463, "F", 19129, "ar theodolites. deposits haggle furiously regular accounts. final courts after",
3797830, "O", 17161, "ffix quickly pending foxes. foxes nag: final\, express requests nag. slyly",
12647, "O", 33822, ". regular theodolites sleep after the care",
1472960, "O", 17528, "furiously after the packages. quickly even id",
3800352, "O", 39159, "uickly above the quickly ironic instructions; even requests about t",
1483047, "O", 37704, "ts use slyly across the idly regular f",
43300, "O", 52391, "s. express requests are. ironic requests are carefully after th",
Showing 10/235398 results !
Table Selection time: 0.045552 seconds
```

c) Query 3: select * from orders where o_totalprice = 52391 => **~ 0.033s**



```
Enter a query (enter 'quit' to quit): select * from orders where o_totalprice = 52391
Parsed successfully!
Table name: orders
select fields[0] = o_orderkey
select fields[1] = o_orderstatus
select fields[2] = o_totalprice
select fields[3] = o_comment
where fields[0] = o_totalprice
where operators[0] = =
value values[0] = 52391
*** Query result ***
o_orderkey, o_orderstatus, o_totalprice, o_comment,
43300, "O", 52391, "s. express requests are. ironic requests are carefully after th",
5539750, "O", 52391, "en foxes! blithely even depths lose",
3082690, "O", 52391, "ly furiously unusual requests. express deposits are stealthily",
872549, "O", 52391, "the packages. ironic\, even dependencies",
1302881, "F", 52391, "re blithely. even\, regular platelets detect about the quickly sil",
5481542, "O", 52391, "nstructions solve carefully even requests. final platelets cajole bli",
3818502, "F", 52391, "lent accounts wake quickly regular instructio",
4771648, "F", 52391, "ajole carefully final pinto beans. quietly ironic platelets are slyly. qu",
4972225, "F", 52391, "r requests haggle after the theodolites. quickly regula",
Showing 9/9 results !
Table Selection time: 0.033199 seconds
```

d) Query 4: select * from orders where o_orderstatus = “P” => **~ 0.018s**



```
Enter a query (enter 'quit' to quit): select * from orders where o_orderstatus = "P"
Parsed successfully!
Table name: orders
select fields[0] = o_orderkey
select fields[1] = o_orderstatus
select fields[2] = o_totalprice
select fields[3] = o_comment
where fields[0] = o_orderstatus
where operators[0] = =
value values[0] = P
*** Query result ***
o_orderkey, o_orderstatus, o_totalprice, o_comment,
5820391, "P", 179356, "nts. regular packages boost depo",
5837701, "P", 240521, "ites run along the bl",
5860608, "P", 253187, "ments wake after the carefully furious packages. furiously even foxes d",
5861281, "P", 166407, "ide of the ironic deposits; express\, unusual dependen",
5862055, "P", 182976, "yly regular warhorses are. blithely",
5863141, "P", 88244, "ke thinly. ironic accounts sleep. pending\, ironic instructions are-- sl",
5982887, "P", 134175, "ckly regular\, express instructions. carefully brave sauternes s",
5987043, "P", 177117, "arefully bold packages sleep. ironic warhorses affix",
4506022, "P", 166188, "ajole furiously regular pinto beans. ironic dolphins nag ab",
4540387, "P", 198608, "e quickly according to the slyly quiet accounts. carefu",
Showing 10/38543 results !
Table Selection time: 0.018333 seconds
```

7. Limitation of this version:

- Has not processed INSERT query -> only works with sample data (orders table).

- Has not supported some operators such as: >= (greater than). <= (less than), <> (not equal), like and not like.