Data Blocks: Hybrid OLTP and OLAP on Compressed Storage using both Vectorization and Compilation

Authors: Harald Lang, Tobias Mühlbauer, Florian Funke,
Peter Boncz, Thomas Neumann, Alfons Kemper
(June 2016, DOI: 10.1145/2882903.2882925, Conference: SIGMOD)

Contributed by:
Group 7
Meet Shah 202011047
Krunal Mehta 202011051

Guided by:
Prof. Minal Bhise
Mayank Patel
Kalgi Gandhi

1. Introduction

Online Transaction Processing (OLTP) databases are designed for online transaction queries like insert, update and delete. An example is banking sector or stock market-related queries where updating is frequent. Online Analytical Processing (OLAP) deals with Archival Data or Historical Data and a very small amount of transactions. The main thing of this experiment is that it talks about hybrid OLTP and OLAP system. Here data set is divided into two parts, hot, which is frequently accessed, and cold, which is infrequently accessed. Data Blocks are essentially compressed cold data chunks [1]. Further, to speed up Data Blocks scans, Positional Small Materialized Aggregates (PSMAs) - a lightweight index structure containing a concise lookup table along with a simple min/max attribute is used. The scan range can be decreased by the PSMA indexes even if the whole Data Block cannot be skipped during a scan.

1.1 Motivation

As data is increasing rapidly, data summarization has to be done to handle large amounts of data efficiently. Data summarization affects primary storage data reduction and will continue to play an important role in data reduction.

1.2 Objective

The main goal is to improve query performance in a hybrid OLAP-OLTP database system by data summarization without affecting query efficiency. For that, division of data into hot and cold data, compression of cold data, and PSMA Lookup Table implementation are the main objectives of the experiment.

1.3 Scope

The experiment is done on Relational Database considering both types of queries, OLTP (Online Transaction Processing) and OLAP (Online Analytical Processing).

2. Data Blocks Flow, Data Structures, and Algorithm

This section elaborates flow, data structures, and algorithm of data blocks technique.

2.1 Flow

Here firstly, we are loading the TPC-H dataset into our system. Then we are dividing Lineitem Table into hot and cold chunks and compressing cold chunks. Further, we are generating PSMA Lookup Table.

Later, after the loading of the queryset, if a query is required to access cold data, we will do Data Blocks scan to the range extracted from PSMA Lookup Table, uncompress and scan it. Otherwise, a scan will be performed on hot uncompressed data.
Lastly, the query will be executed and the output will be shown.

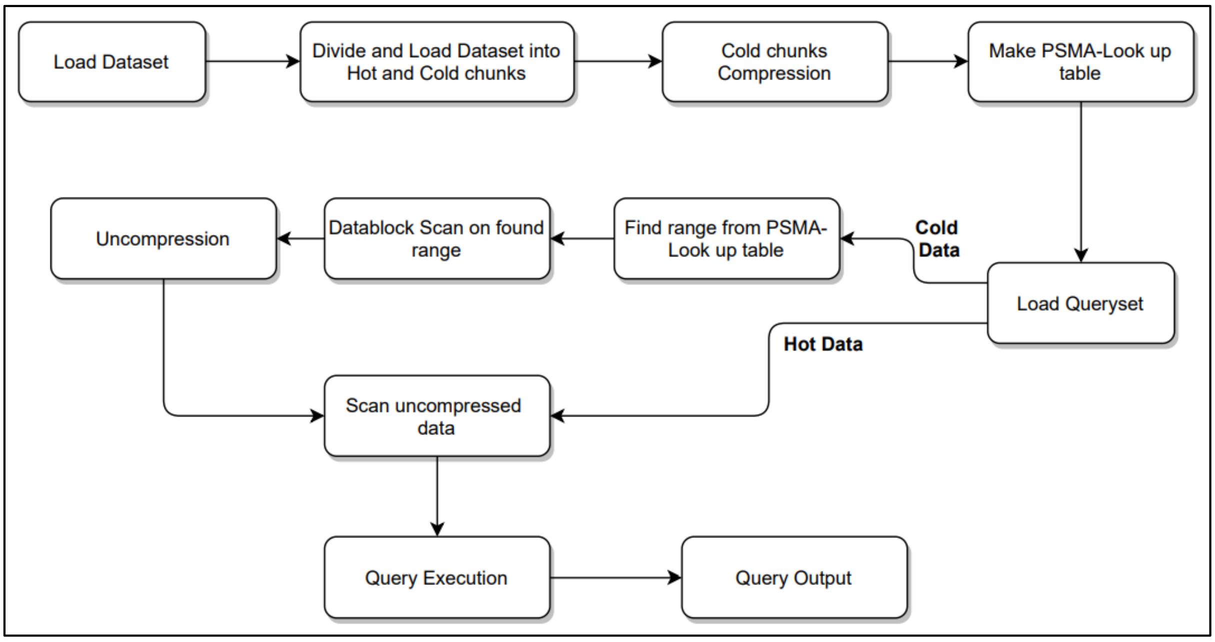


Fig.1. Flow of Data Blocks

2.2 Data Structures

In this experiment, a data-centric approach is used. Firstly Lineitem Table is sorted on the l_shipdate attribute, then 0.2M tuples were extracted from the starting date specified in the Key-Value List.

Further tuples are inserted into Lineitem_Hot Table and Lineitem_Cold Table according to the limit and offset values provided by Key-Value List.
For compression, truncation compression technique is applied on the l_shipdate attribute of Lineitem Cold Table.

PSMA Lookup Table will contain offset, limit, and no_of_records for each index.

Table1. Key-Value List

min_cold	0
max_cold	65536
min_hot	65536
max_hot	200000
partition_start_date	1992-01-02

Table2. Lineitem_Cold Table (Compressed)

l_orderkey	l_partkey	l_suppkey	l_linenumber	l_quantity	l_extendedprice	l_discount	l_tax	l_returnflag	l_linestatus
89173	4190	5	17	19756.89	0.04	0.01	A	F	89173
148071	3100	5	29	32453.03	0.05	0.06	A	F	148071
176498	1533	5	16	25191.84	0.06	0.02	R	F	176498
64313	1832	5	37	47260.47	0.06	0.02	R	F	64313
4641	4642	2	6	9273.84	0	0.08	A	F	4641

l_shipdate	l_commitdate	l_receiptdate	l_shipinstruct	l_shipmode	l_comment
1	1994-12-26	1995-01-01	DELIVER IN PERSON	REG AIR	inal packages haggle carefully
11	1994-11-17	1995-02-04	COLLECT COD	MAIL	l requests hagg
12	1994-12-27	1995-01-12	DELIVER IN PERSON	SHIP	kages cajole carefully
14	1994-12-29	1995-01-31	DELIVER IN PERSON	FOB	deas use blithely! special foxes print af
19	1994-12-08	1995-02-10	NONE	TRUCK	sits wake furiously regular

Table3. Lineitem_Hot Table

l_orderkey	l_partkey	l_suppkey	l_linenumber	l_quantity	l_extendedprice	l_discount	l_tax	l_returnflag	l_linestatus
359	11158	6161	2	18	19244.7	0	0.03	A	F
391	121586	6611	1	14	22506.12	0.09	0.02	R	F
12101	1596	1597	2	25	37439.75	0.04	0.04	A	F
930	99635	2145	4	21	34327.23	0.06	0.02	A	F
10722	113068	5580	6	6	6486.36	0.09	0.03	R	F

l_shipdate	l_commitdate	l_receiptdate	l_shipinstruct	l_shipmode	l_comment
1995-01-27	1995-03-18	1995-01-31	DELIVER IN PERSON	RAIL	unusual warthogs. ironically sp
1995-02-11	1995-02-03	1995-02-13	TAKE BACK RETURN	TRUCK	escapades sleep furiously about
1995-02-13	1995-03-02	1995-02-14	COLLECT COD	REG AIR	al, final foxes about the ir
1995-02-16	1995-03-03	1995-03-13	DELIVER IN PERSON	SHIP	foxes. regular deposits integrate carefu
1995-02-12	1995-04-02	1995-02-21	NONE	FOB	eas. carefully special deposits after the

Table4. PSMA Lookup Table

index	offset	limit	no_of_records
0	0	2497	2497
1	2497	4979	2482
2	4979	7470	2491
3	7470	9910	2440
4	9910	12506	2596

2.3 Algorithm

Below algorithm is the step by step process in order to execute query by Data Blocks technique for hybrid OLAP-OLTP system.

```
1. Start
2. Load Dataset
3. Initialize variables according to Data Structures
4. Divide data into hot data and cold data
   a. Sort Lineitem Table on l_shipdate attribute
   b. Insert data into Lineitem_Cold Table and Lineitem_Hot Table according to Data
      Structures variables
5. Compress cold data using truncation compression scheme
   a. Extract min and max of l_shipdate from Lineitem_Cold Table
   b. Calculate and store delta value of l_shipdate for all tuples by subtracting min value
      from it
6. Make a PSMA Lookup Table for cold data
   a. Find number of entry in PSMA Lookup Table by subtracting min l_shipdate from
      max l_shipdate
   b. Insert index of each entry as delta value
   c. Insert offset value of each entry by calculating number of records of all previous
      entries
   d. Insert number_of_records value of each entry by calculating number of tuples
      (of that entry) present in Lineitem_Cold Table
   e. Insert limit value of each entry by addition of offset value and number_of_records
      value
7. Load queryset
8. For all query in queryset do
      If Data Block = cold
            Find out range from PSMA Lookup Table
            Data Block scan on found range
            Uncompress data
            Scan uncompressed data
      Else
            Scan uncompressed data
      End if
      Query Output
   End-for
9. Stop
```

3. Experimental Setup

This section contains details regarding the Dataset and Queryset used in the experiment, Software and Hardware specification for the execution of the experiment, and Evaluation Parameters of the experiment.

3.1 Dataset and Queryset

Benchmark TPC-H dataset is a decision support benchmark, it consists of a set of business oriented queries [2]. Size of the dataset used here is 21 MB which comprises 0.2M tuples.

3.2 Software and Hardware Setup

Experiments are performed using a system with 8 GB memory, core i5 processor, with a hard disk of 2TB. PostgreSQL 10.14 and Python 3.9 are used as software for the implementation [3, 4]. Additionally psycopg2 and matplotlib modules were used in Python in order to connect with PostgreSQL and to plot graphs respectively.

3.3 Evaluation Parameters

Evaluation Parameters for the experiment are identified as Algorithm Execution Time (AET) which calculates execution time of each algorithm. Execution time of each query is depicted as Query Execution Time (QET).

3.4 Block Diagram

Block Diagram of the experiment is shown in Fig.2.

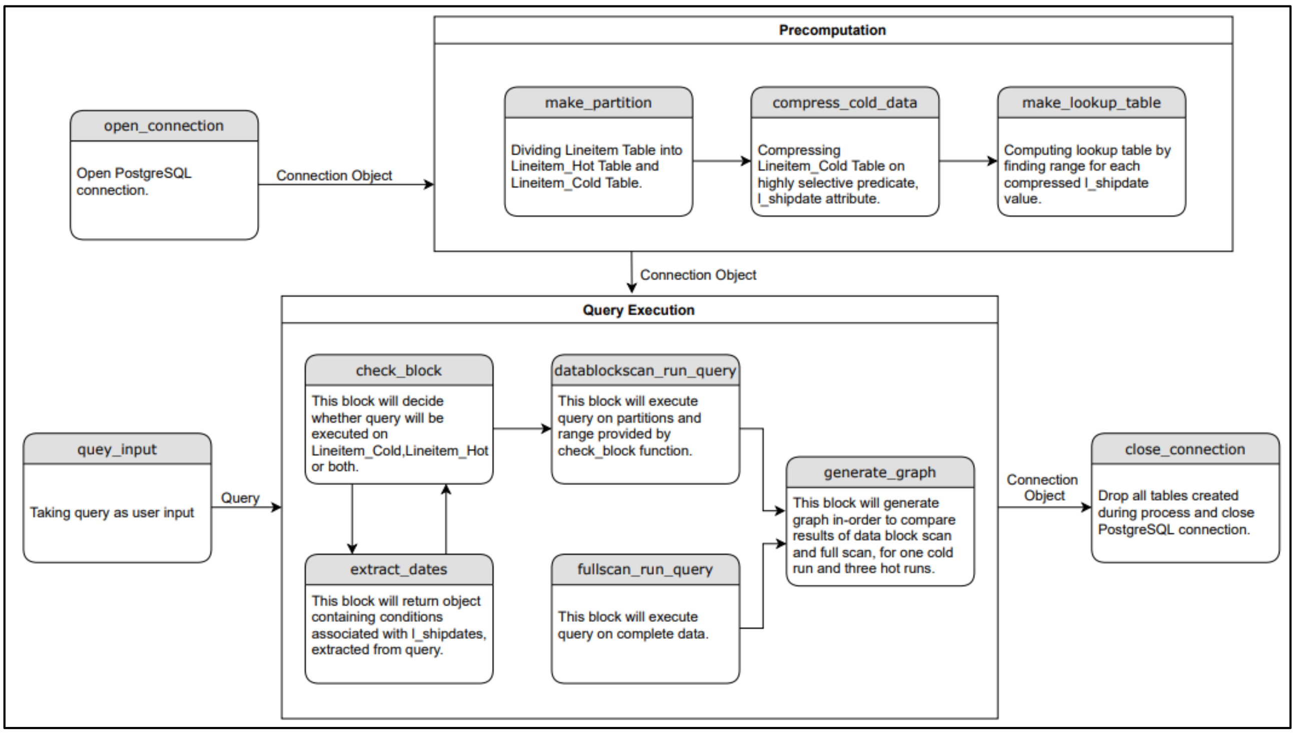

Precomputation
open_connection
Open PostgreSQL connection.
Connection Object
make_partition
Dividing Lineitem Table into Lineitem_Hot Table and Lineitem_Cold Table.
compress_cold_data
Compressing Lineitem_Cold Table on highly selective predicate, l_shipdate attribute.
make_lookup_table
Computing lookup table by finding range for each compressed l_shipdate value.
Connection Object
Query Execution
quey_input
Taking query as user input
Query
check_block
This block will decide whether query will be executed on Lineitem_Cold,Lineitem_Hot or both.
datablockscan_run_query
This block will execute query on partitions and range provided by check_block function.
extract_dates
This block will return object containing conditions associated with l_shipdates, extracted from query.
fullscan_run_query
This block will execute query on complete data.
generate_graph
This block will generate graph in-order to compare results of data block scan and full scan, for one cold run and three hot runs.
Connection Object
close_connection
Drop all tables created during process and close PostgreSQL connection.

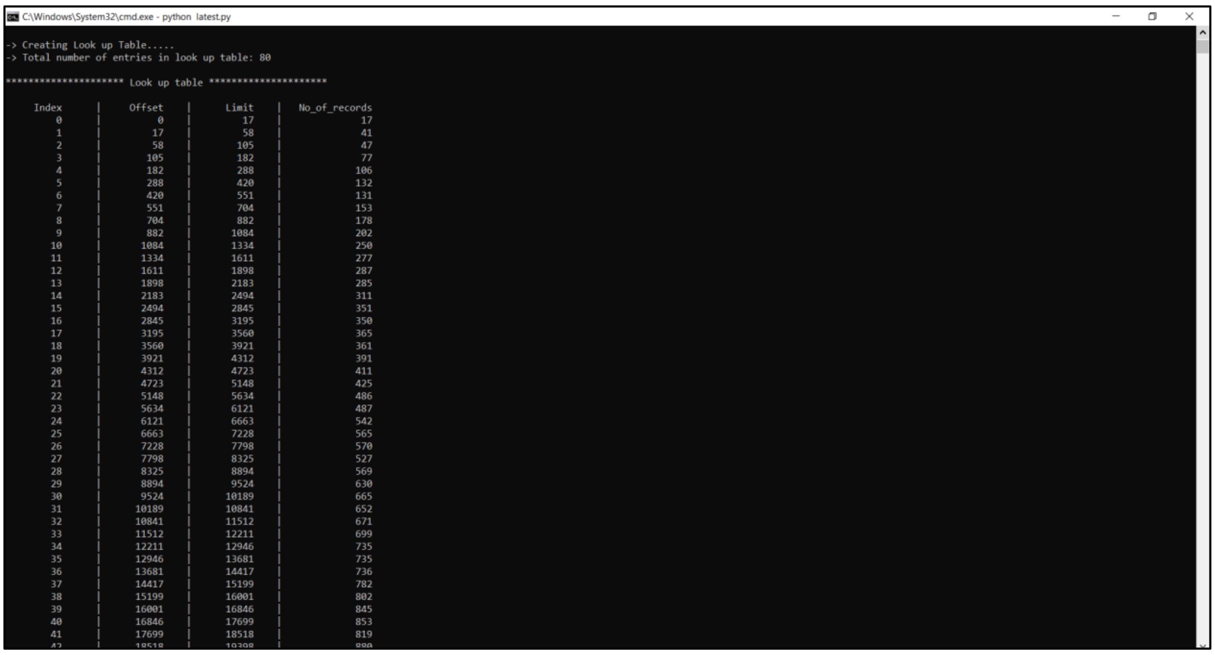
Fig.2. Block Diagram

4. Results and Discussions

This section contain the results obtain from the experiment.

4.1 PSMA Lookup Table Implementation

PSMA Lookup Table containing offset, limit, and no_of_records for each delta value as index is depicted in Fig.3.

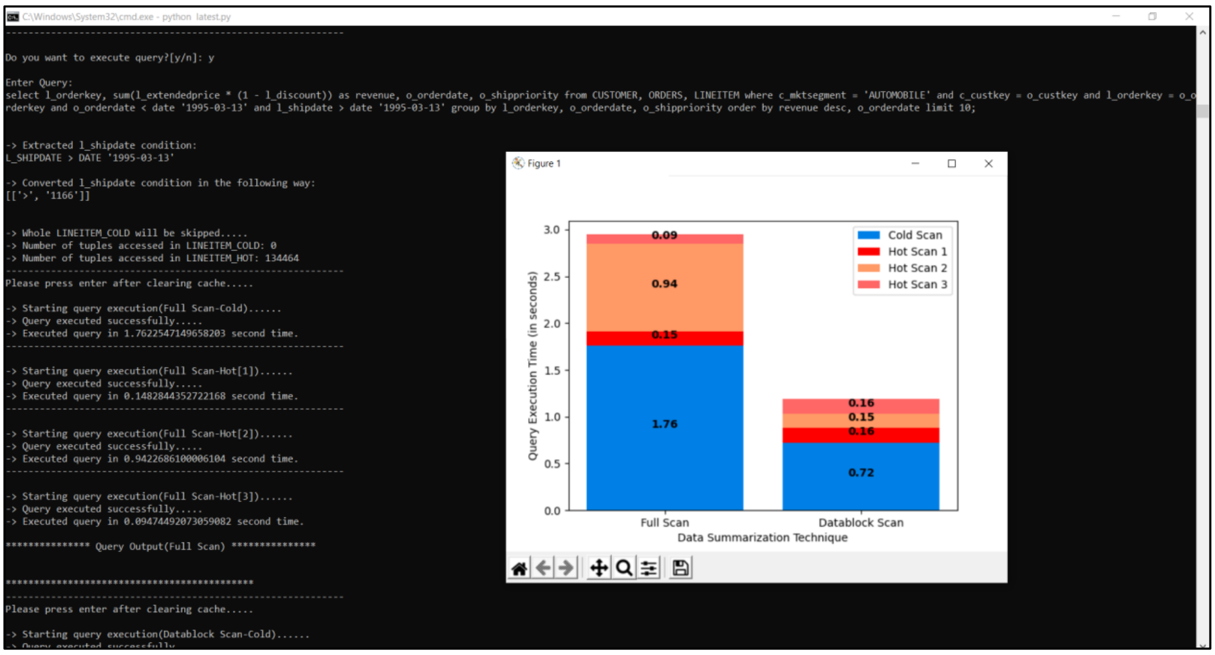


```
C:\Windows\System32\cmd.exe - python  latest.py

-> Creating Look up Table.....
-> Total number of entries in look up table: 80

******************** Look up table ********************

     Index      |     Offset     |     Limit      |   No_of_records
        0       |           0    |         17     |              17
        1       |          17    |         58     |              41
        2       |          58    |        105     |              47
        3       |         105    |        182     |              77
        4       |         182    |        288     |             106
        5       |         288    |        420     |             132
        6       |         420    |        551     |             131
        7       |         551    |        704     |             153
        8       |         704    |        882     |             178
        9       |         882    |       1084     |             202
       10       |        1084    |       1334     |             250
       11       |        1334    |       1611     |             277
       12       |        1611    |       1898     |             287
       13       |        1898    |       2183     |             285
       14       |        2183    |       2494     |             311
       15       |        2494    |       2845     |             351
       16       |        2845    |       3195     |             350
       17       |        3195    |       3560     |             365
       18       |        3560    |       3921     |             361
       19       |        3921    |       4312     |             391
       20       |        4312    |       4723     |             411
       21       |        4723    |       5148     |             425
       22       |        5148    |       5634     |             486
       23       |        5634    |       6121     |             487
       24       |        6121    |       6663     |             542
       25       |        6663    |       7228     |             565
       26       |        7228    |       7798     |             570
       27       |        7798    |       8325     |             527
       28       |        8325    |       8894     |             569
       29       |        8894    |       9524     |             630
       30       |        9524    |      10189     |             665
       31       |       10189    |      10841     |             652
       32       |       10841    |      11512     |             671
       33       |       11512    |      12211     |             699
       34       |       12211    |      12946     |             735
       35       |       12946    |      13681     |             735
       36       |       13681    |      14417     |             736
       37       |       14417    |      15199     |             782
       38       |       15199    |      16001     |             802
       39       |       16001    |      16846     |             845
       40       |       16846    |      17699     |             853
       41       |       17699    |      18518     |             819
       42       |       18518    |      19398     |             880
```

Fig.3. PSMA Lookup Table

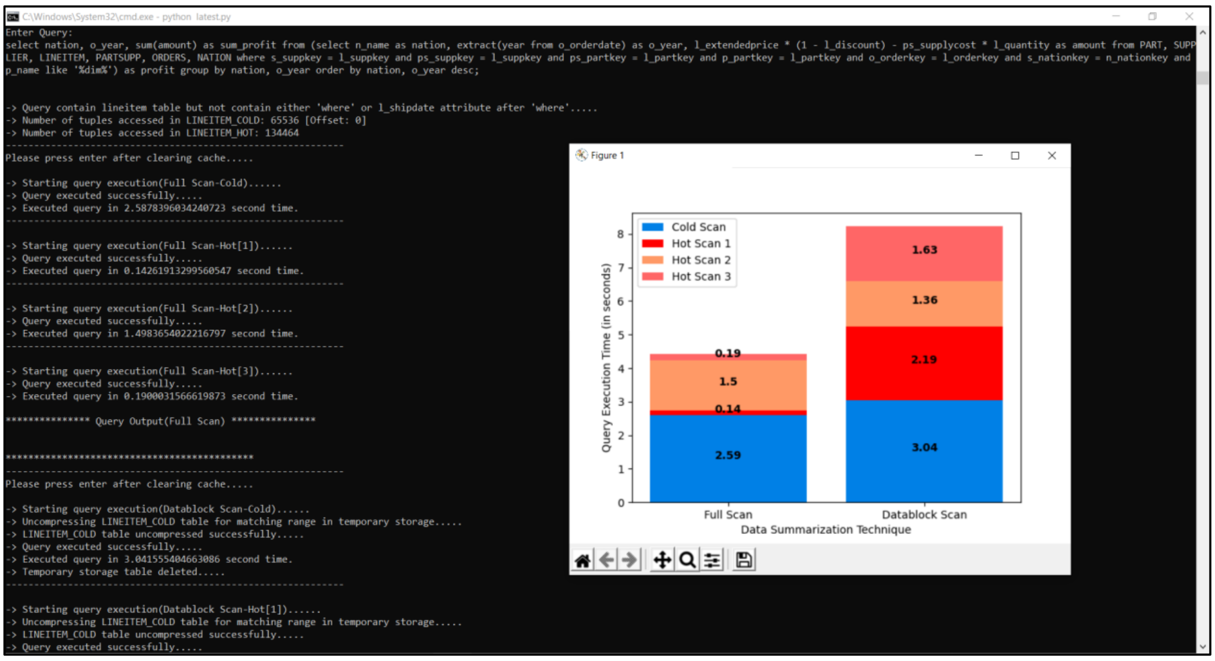
4.2 Query Execution

Query execution with Data Block Scan algorithm, and comparison with Full Scan is shown in Fig.4, Fig.5 and Fig.6.



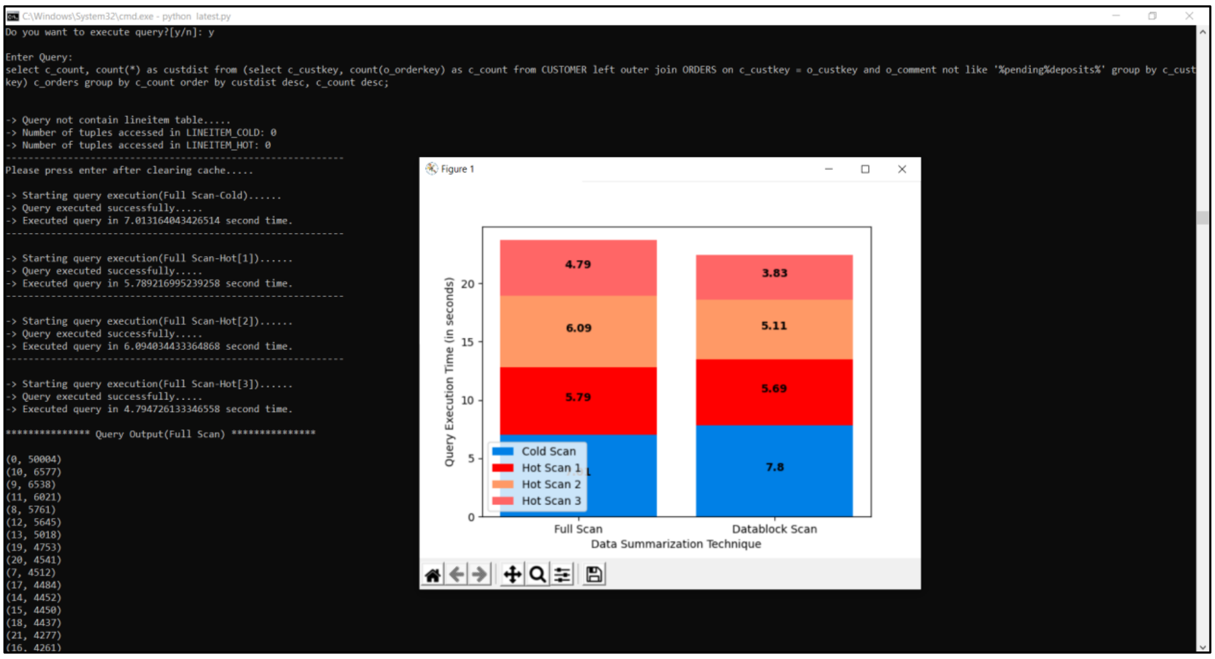
```
C:\Windows\System32\cmd.exe - python  latest.py
------------------------------------------------------------

Do you want to execute query?[y/n]: y

Enter Query:
select l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue, o_orderdate, o_shippriority from CUSTOMER, ORDERS, LINEITEM where c_mktsegment = 'AUTOMOBILE' and c_custkey = o_custkey and l_orderkey = o_o
rderkey and o_orderdate < date '1995-03-13' and l_shipdate > date '1995-03-13' group by l_orderkey, o_orderdate, o_shippriority order by revenue desc, o_orderdate limit 10;


-> Extracted l_shipdate condition:
L_SHIPDATE > DATE '1995-03-13'

-> Converted l_shipdate condition in the following way:
[['>', '1166']]


-> Whole LINEITEM_COLD will be skipped.....
-> Number of tuples accessed in LINEITEM_COLD: 0
-> Number of tuples accessed in LINEITEM_HOT: 134464
------------------------------------------------------------
Please press enter after clearing cache.....

-> Starting query execution(Full Scan-Cold)......
-> Query executed successfully.....
-> Executed query in 1.7622547149658203 second time.
------------------------------------------------------------

-> Starting query execution(Full Scan-Hot[1])......
-> Query executed successfully.....
-> Executed query in 0.1482844352722168 second time.
------------------------------------------------------------

-> Starting query execution(Full Scan-Hot[2])......
-> Query executed successfully.....
-> Executed query in 0.9422686100006104 second time.
------------------------------------------------------------

-> Starting query execution(Full Scan-Hot[3])......
-> Query executed successfully.....
-> Executed query in 0.09474492073059082 second time.

*************** Query Output(Full Scan) ***************


********************************************
------------------------------------------------------------
Please press enter after clearing cache.....

-> Starting query execution(Datablock Scan-Cold)......
-> Query executed successfully.....
```

Fig.4. Query Execution (OLAP-Q3)



```
Enter Query:
select nation, o_year, sum(amount) as sum_profit from (select n_name as nation, extract(year from o_orderdate) as o_year, l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount from PART, SUPP
LIER, LINEITEM, PARTSUPP, ORDERS, NATION where s_suppkey = l_suppkey and ps_suppkey = l_suppkey and ps_partkey = l_partkey and p_partkey = l_partkey and o_orderkey = l_orderkey and s_nationkey = n_nationkey and
p_name like '%dim%') as profit group by nation, o_year order by nation, o_year desc;


-> Query contain lineitem table but not contain either 'where' or l_shipdate attribute after 'where'.....
-> Number of tuples accessed in LINEITEM_COLD: 65536 [Offset: 0]
-> Number of tuples accessed in LINEITEM_HOT: 134464
------------------------------------------------------------
Please press enter after clearing cache.....

-> Starting query execution(Full Scan-Cold)......
-> Query executed successfully.....
-> Executed query in 2.5878396034240723 second time.
------------------------------------------------------------

-> Starting query execution(Full Scan-Hot[1])......
-> Query executed successfully.....
-> Executed query in 0.14261913299560547 second time.
------------------------------------------------------------

-> Starting query execution(Full Scan-Hot[2])......
-> Query executed successfully.....
-> Executed query in 1.4983654022216797 second time.
------------------------------------------------------------

-> Starting query execution(Full Scan-Hot[3])......
-> Query executed successfully.....
-> Executed query in 0.1900031566619873 second time.

*************** Query Output(Full Scan) ***************


********************************************
------------------------------------------------------------
Please press enter after clearing cache.....

-> Starting query execution(Datablock Scan-Cold)......
-> Uncompressing LINEITEM_COLD table for matching range in temporary storage.....
-> LINEITEM_COLD table uncompressed successfully.....
-> Query executed successfully.....
-> Executed query in 3.041555404663086 second time.
-> Temporary storage table deleted.....
------------------------------------------------------------

-> Starting query execution(Datablock Scan-Hot[1])......
-> Uncompressing LINEITEM_COLD table for matching range in temporary storage.....
-> LINEITEM_COLD table uncompressed successfully.....
-> Query executed successfully.....
```

Fig.5. Query Execution (OLAP-Q9)


C:\Windows\System32\cmd.exe - python latest.py
Figure 1
Cold Scan
Hot Scan 1
Hot Scan 2
Hot Scan 3
Query Execution Time (in seconds)
0
5
10
15
20
4.79
6.09
5.79
3.83
5.11
5.69
7.8
Full Scan
Datablock Scan
Data Summarization Technique

```
Do you want to execute query?[y/n]: y

Enter Query:
select c_count, count(*) as custdist from (select c_custkey, count(o_orderkey) as c_count from CUSTOMER left outer join ORDERS on c_custkey = o_custkey and o_comment not like '%pending%deposits%' group by c_cust
key) c_orders group by c_count order by custdist desc, c_count desc;


-> Query not contain lineitem table.....
-> Number of tuples accessed in LINEITEM_COLD: 0
-> Number of tuples accessed in LINEITEM_HOT: 0
------------------------------------------------------------
Please press enter after clearing cache.....

-> Starting query execution(Full Scan-Cold)......
-> Query executed successfully.....
-> Executed query in 7.013164043426514 second time.
------------------------------------------------------------

-> Starting query execution(Full Scan-Hot[1])......
-> Query executed successfully.....
-> Executed query in 5.789216995239258 second time.
------------------------------------------------------------

-> Starting query execution(Full Scan-Hot[2])......
-> Query executed successfully.....
-> Executed query in 6.094034433364868 second time.
------------------------------------------------------------

-> Starting query execution(Full Scan-Hot[3])......
-> Query executed successfully.....
-> Executed query in 4.794726133346558 second time.

*************** Query Output(Full Scan) ***************

(0, 50004)
(10, 6577)
(9, 6538)
(11, 6021)
(8, 5761)
(12, 5645)
(13, 5018)
(19, 4753)
(20, 4541)
(7, 4512)
(17, 4484)
(14, 4452)
(15, 4450)
(18, 4437)
(21, 4277)
(16, 4261)
```

Fig.6. Query Execution (OLAP-Q13)

4.3 Experiment Results

AET of the Data Blocks experiment is recorded as 342.96 seconds.
For each query, algorithm is executed once for cold run and thrice for hot run, for both Full Scan and Data Block Scan.

4.3.1 Data Summarization (OLAP)

QET for OLAP is depicted in Fig.7, which shows average QET for both Full Scan and Data Block Scan for all 22 OLAP TPC-H benchmark queries.

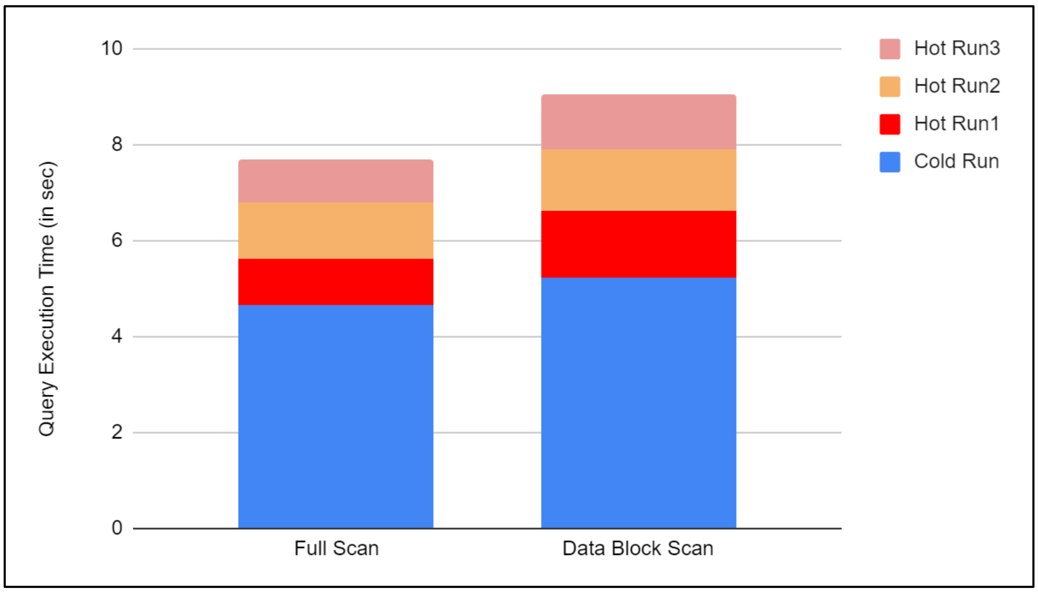


Fig.7. Data Summarization (OLAP)

4.3.2 Data Summarization (OLTP)

QET for OLTP is depicted in Fig.8, which shows average QET for both Full Scan and Data Block Scan for all 22 modified OLTP TPC-H queries.

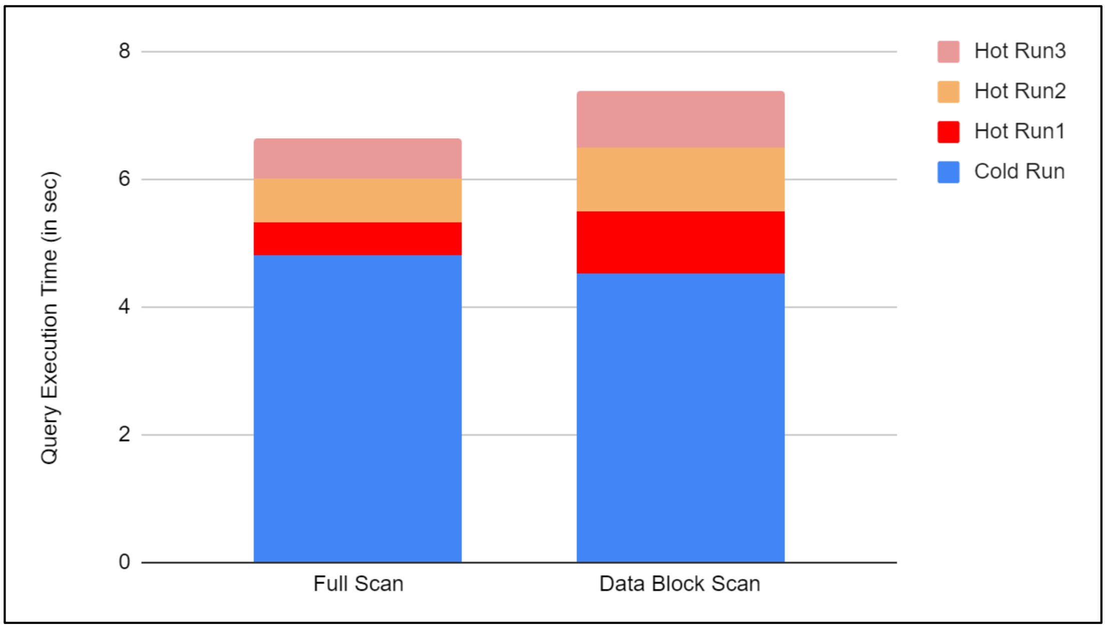


Fig.8. Data Summarization (OLTP)

5. Conclusion

We observe that when a query has to access Lineitem_Cold Table for evaluation, its overhead increase due to first uncompressing required cold data and then executing a query on that, but in the case when the query does not need to access Lineitem_Cold for evaluation it performs better as it skips Lineitem_Cold Table data and only executes a query on Lineitem_Hot Table data.

6. References

[1] H. Lang, T. Muhlbauer, F. Funke, P. Boncz, T. Neumann, A. Kemper, Data Blocks: Hybrid OLTP and OLAP on Compressed Storage using both Vectorization and Compilation, SIGMOD, 2016, pp. 311-326.

[2] TPC-H Dataset and Queryset: http://www.tpc.org/tpch/

[3] Python: https://www.python.org/downloads/windows/

[4] PostgreSQL: https://www.postgresql.org/download/