



On-Disk Bitmap Index Performance in Bizgres 0.9

A Greenplum Whitepaper

April 2, 2006

Author: **Ayush Parashar**
Performance Engineering Lab

Table of Contents

1.0	Summary	1
2.0	Introduction.....	1
3.0	Performance Test Configuration.....	2
3.1	Database Schema.....	2
3.2	Hardware Configuration.....	3
4.0	Index Creation Results.....	3
5.0	Query Performance Results.....	5
5.1	Queries Used for the Performance Test.....	6
5.1.1	Query 1.....	6
5.1.2	Query 2.....	7
5.1.3	Query 3.....	7
5.1.4	Query 4.....	7
5.1.5	Query 5.....	7
6.0	Conclusion	7
Appendix A: Table Definitions.....		8
	LINEITEM.....	8
	ORDERS.....	8
	CUSTOMER.....	9
Appendix B: Query Plans.....		10
	QUERY 1 – BITMAP.....	10
	QUERY 1 – B-TREE.....	11
	QUERY 2 – BITMAP.....	13
	QUERY 2 – B-TREE.....	14
	QUERY 3 – BITMAP.....	14
	QUERY 3 – B-TREE.....	15
	QUERY 4 – BITMAP.....	15
	QUERY 4 – B-TREE.....	16
	QUERY 5 – BITMAP.....	17
	QUERY 5 – B-TREE.....	17

List of Tables and Figures

Table 1: Indexes Created for Index Performance Test	2
Table 2: Hardware Specifications Used for Index Performance Test	3
Table 3: Index Creation Times and Sizes	3
Table 4: Query Response Times Using Bitmap vs. B-tree Indexes	5
Figure 1: Index Creation Times for Bitmap vs. Btree Indexes	4
Figure 2: Index Size for Bitmap vs. Btree Indexes.....	4
Figure 3: Query Response Times Using Bitmap vs. B-tree Indexes	5

1.0 Summary

Bizgres 0.9 introduces a powerful new data-warehousing feature – on-disk bitmap index. This paper demonstrates the benefits offered by bitmap indexes over traditional b-tree indexes in terms of space used, index creation time, and query performance.

2.0 Introduction

On-disk bitmap index (also referred to as *bitmap index*) was introduced in the Bizgres 0.9 release. Bitmap index is an access method that is specifically suited for data warehousing. A regular index, such as a b-tree, stores a list of tuple ids for each key corresponding to the rows with that key value. In a bitmap index, a bitmap for each key value replaces the list of tuple ids. These bitmaps are stored in the bitmap index in a compressed format making it far more efficient in terms of space requirements and index creation time, especially on low-cardinality columns of high-dimension fact tables. Bitmap indexes not only offer benefits during index creation, but also in query performance. The use of bitmap indexes in data warehousing environments can dramatically improve response times for large classes of ad hoc queries.

Bitmap indexes are most effective for queries that contain multiple conditions in the `WHERE` clause. `AND` and `OR` conditions in the `WHERE` clause of a query can be resolved quickly by performing the corresponding Boolean operations directly on the bitmaps before converting the resulting bitmap to tuple ids. If the resulting number of rows is small, the query can be answered quickly without resorting to a full table scan. It should be noted that benefits of bitmap index are especially significant when the index is on a column with low cardinality.

This paper demonstrates and quantifies these performance gains. Data derived from TPC-H™ examples¹ was used due to its popularity and unaltered and unbiased data generation. The schema and SQL queries used for these performance tests are documented in this paper. Although results are shown for queries derived from the TPC-H™ data, the benefits of bitmap indexes extend to real-world decision support systems and data warehouses. Customers are encouraged to experiment with this feature to gauge its benefits.

¹ TPC™ is a registered trademark of the Transaction Processing Performance Council. The workload used in this whitepaper is derived from the TPC-H™ examples available from www.tpc.org. This paper makes no comparisons or references to TPC-H™ benchmarks.

3.0 Performance Test Configuration

For the results documented in this paper, two identical Bizgres 0.9 databases were used. Both databases used the same hardware, storage, schema, data, and server configuration parameters. Default PostgreSQL parameter settings were used for the server runtime configuration.

3.1 Database Schema

The Transaction Processing Performance Council (TPC) is a third-party organization that provides database tools and benchmarks for the industry.

TPC-H™ is their ad-hoc, decision support benchmark and toolset. The schema and data used for the performance tests documented in this paper are derived from the TPC-H™ examples available at www.tpc.org, however this paper makes no comparisons or references to TPC-H™ benchmarks.

Data was generated using the TPC-H™ *dbgen* program with a scale factor of 10. Indexes were created on the following tables:

- LINEITEM (60 million rows)
- ORDERS (15 million rows)
- CUSTOMER (1.5 million rows)

The following table lists the columns on which indexes have been created:

Table 1: Indexes Created for Index Performance Test

Table Name	Indexed Column	Data Type	Cardinality
LINEITEM	L_SHIPMODE	character(10)	7
LINEITEM	L_QUANTITY	numeric(15,2)	50
LINEITEM	L_LINENUMBER	integer	7
LINEITEM	L_SHIPMODE, L_QUANTITY <i>* multi-column index</i>	character(10), numeric(15,2)	350
ORDERS	O_ORDERSTATUS	character(1)	3
ORDERS	O_ORDERPRIORITY	character(15)	5
CUSTOMER	C_MKTSEGMENT	character(10)	5
CUSTOMER	C_NATIONKEY	integer	25

3.2 Hardware Configuration

The following hardware configuration was used:

Table 2: Hardware Specifications Used for Index Performance Test

Processor	Dual CPU AMD Opteron™, 2.2 GHz
RAM	2 GB
OS	64-bit Red Hat Enterprise Linux WS release 3 (Taroon Update 4)
File System	ext2
Storage	JBOD SCSI array with Linux software RAID5

4.0 Index Creation Results

This section provides the results for index creation time and index size for bitmap versus b-tree indexes. The details of the indexed columns are described in [Table 1](#).

The following table shows both the time (in seconds) to create the index and the size of the index (in megabytes) on disk.

Table 3: Index Creation Times and Sizes

#	Indexed Columns	Create Time (seconds)		Space Used (MBs)	
		BITMAP	BTREE	BITMAP	BTREE
1	L_SHIPMODE	454.8	2217.1	58	1804
2	L_QUANTITY	547.2	937.8	117	1804
3	L_LINENUMBER	374.5	412.4	59	1285
4	L_SHIPMODE, L_QUANTITY	948.7	2933.4	176	2845
5	O_ORDERSTATUS	83.5	241.3	5	321
6	O_ORDERPRIORITY	108.5	679.1	11	580
7	C_MKTSEGMENT	10.9	51.3	1	45
8	C_NATIONKEY	8.3	9.3	2	32

The following graphs further illustrate the significant time and storage savings of using bitmap indexes as opposed to b-tree indexes, depending on the cardinality and data type of the indexed column. The bitmap index takes a very small fraction of time and space as compared to b-tree index.

Figure 1: Index Creation Times for Bitmap vs. Btree Indexes

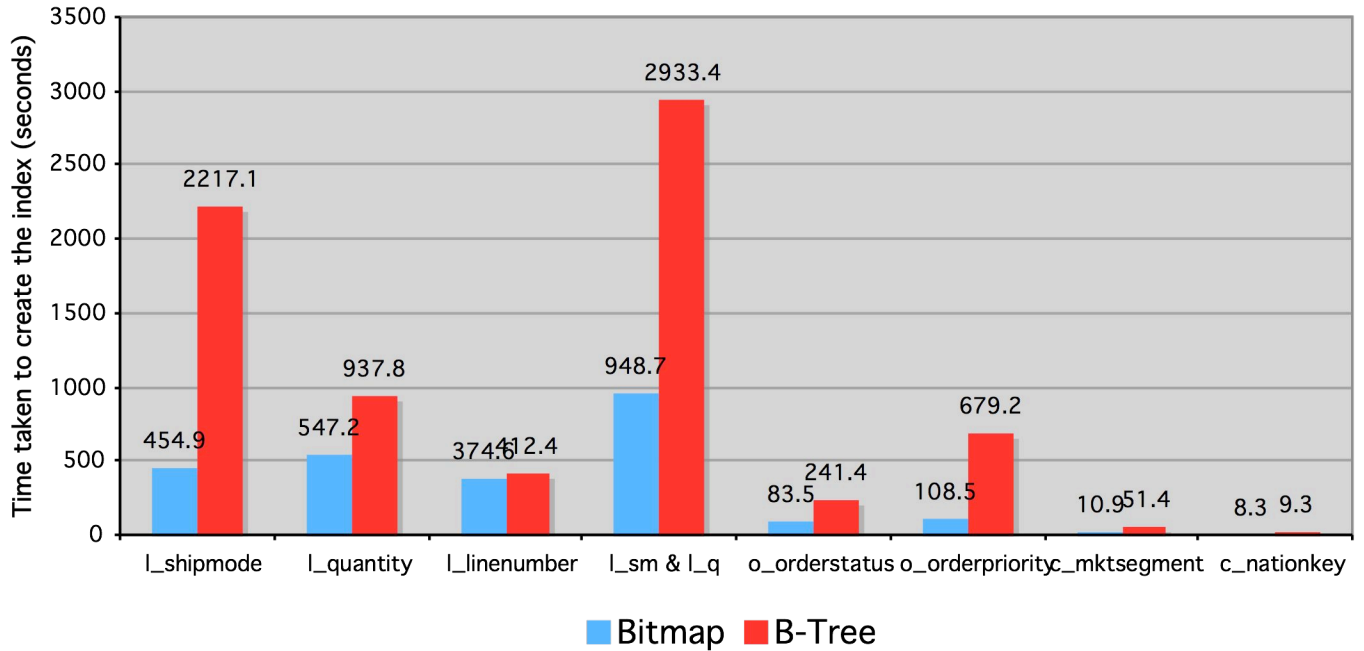
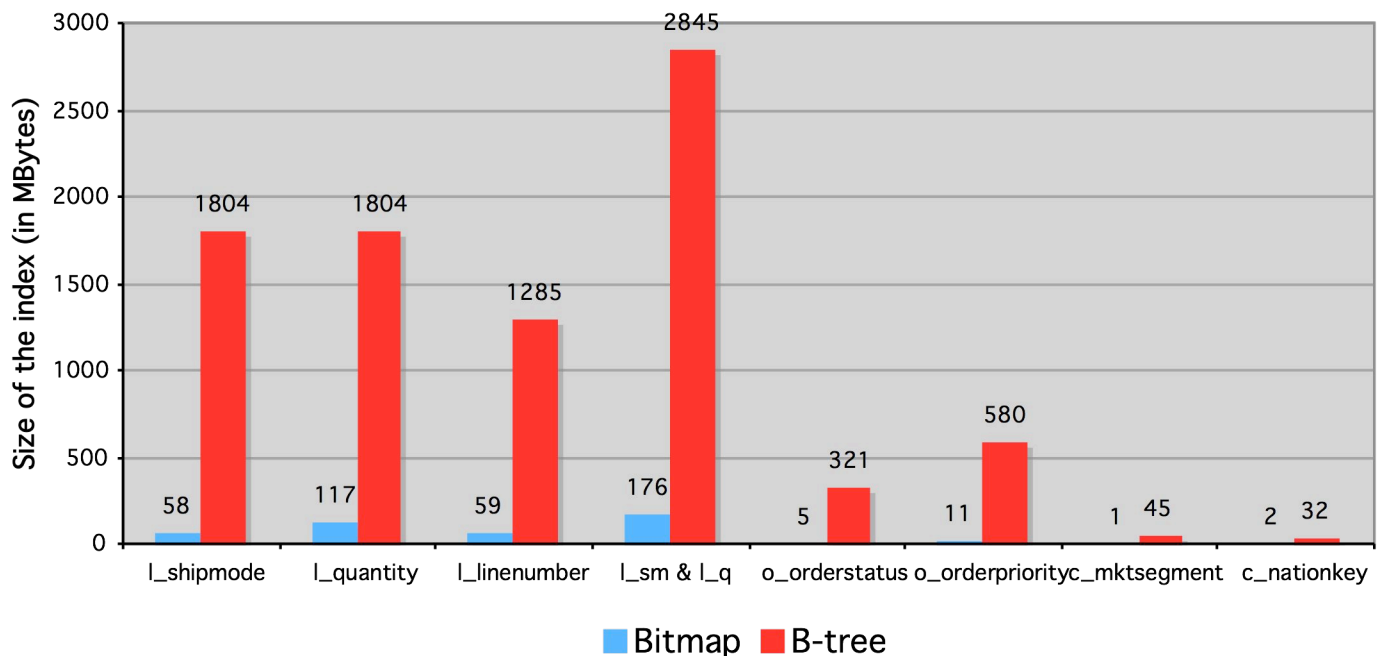


Figure 2: Index Size for Bitmap vs. Btree Indexes



5.0 Query Performance Results

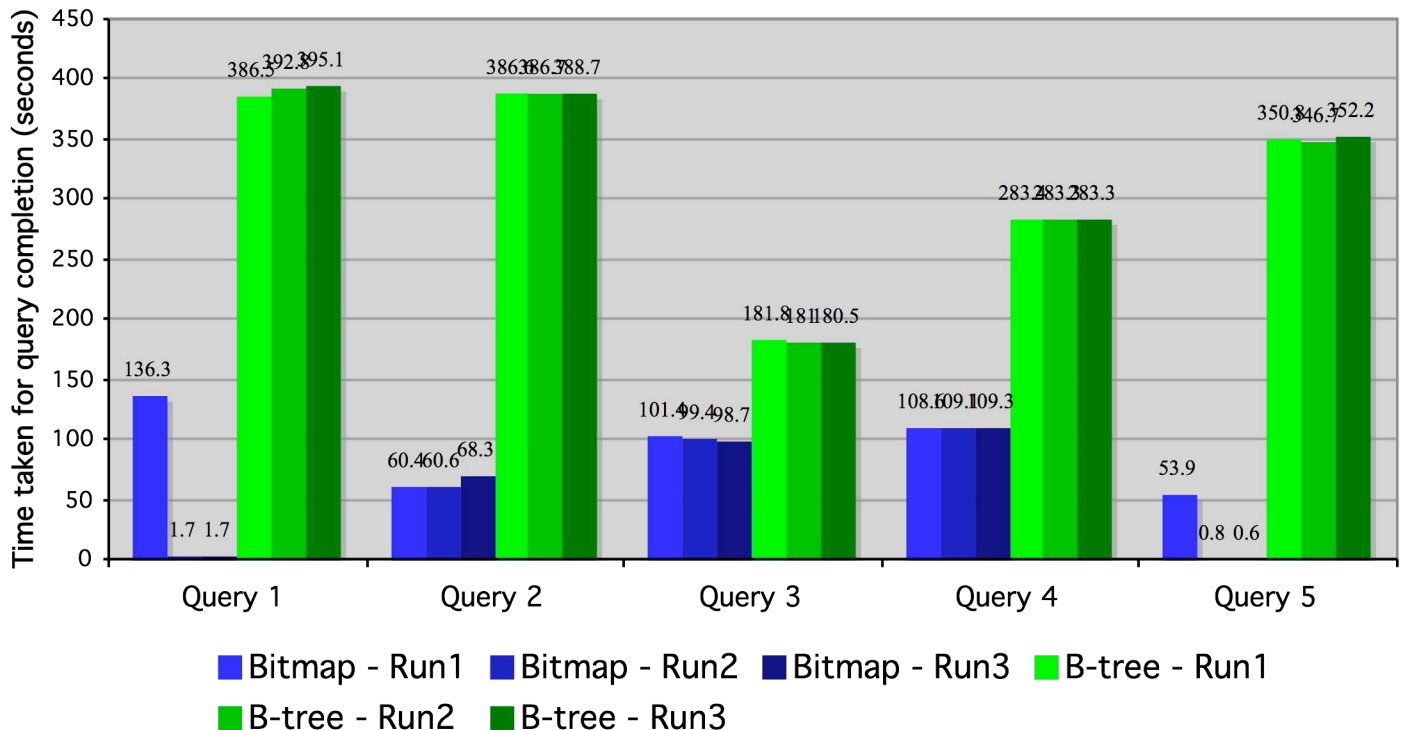
In a data warehouse environment, queries typically access the entire table, include joins of a large table (referred to as a *fact* table) with other small tables (referred to as *dimension* tables), and have some form of aggregation. The use of bitmap indexes can often improve query performance of such typical decision support queries.

The following table shows the query performance (in seconds) for five queries. Results are shown for two databases, one indexed with bitmap indexes and the other with b-tree indexes. Also, results are for various runs of the queries, query being run for three times consecutively.

Table 4: Query Response Times Using Bitmap vs. B-tree Indexes

Query #	Bitmap Index Response Time (in seconds)			B-tree Index Response Time (in seconds)		
	Run1	Run2	Run3	Run1	Run2	Run3
1	136.3	1.7	1.7	386.5	392.8	395.1
2	60.4	60.6	68.3	386.6	386.7	388.7
3	101.4	99.4	98.7	181.8	181	180.5
4	108.6	109.1	109.3	283.4	283.3	283.3
5	53.9	0.8	0.6	350.8	346.7	352.2

Figure 3: Query Response Times Using Bitmap vs. B-tree Indexes



As seen in the graph, query performance improves dramatically when bitmap index is used as compared to b-tree index.

Please refer to the query plans in Appendix B. The gain in performance can be attributed to 'Bitmap And' and 'Bitmap Or' operations being more efficient for bitmap indexes as compared to b-trees. Note that both bitmap and b-tree indexes use the bitmap index scan access method, however the behavior is different for each. With a b-tree index, the b-tree indexes are scanned to create a temporary in-memory bitmap index. With the Bizgres on-disk bitmap index, the bitmap scan retrieves several on-disk bitmap pages at once and provides them to the 'Bitmap And' and 'Bitmap Or' operators. The smaller size of the bitmap indexes combined with the efficiency of the And and Or operators are the reason for the query speed increases.

Another thing to note is that for query 1 and query 5, the time taken for second and third run is dramatically lower than the first run. Since the bitmap indexes are so much smaller than the b-tree indexes, more room is left in memory to store the data accessed from the tables involved in these queries, so that the next time the query is run the table data is in the database and OS cache.

5.1 Queries Used for the Performance Test

This section documents the SQL queries used for the performance test. Queries 1-2 are complex queries with multiple table joins. Queries 3-5 are simple queries.

5.1.1 Query 1

```
SELECT sum(lineitem.l_discount)
FROM
lineitem, orders, customer, nation
WHERE
nation.n_name='UNITED STATES' AND
customer.c_mktsegment='AUTOMOBILE' AND
orders.o_orderstatus='P' AND
orders.o_orderpriority='2-HIGH' AND
lineitem.l_quantity=5 AND
lineitem.l_shipmode='AIR' AND
lineitem.l_linenumber=5 AND
customer.c_custkey=orders.o_custkey AND
orders.o_orderkey=lineitem.l_orderkey AND
nation.n_nationkey=customer.c_nationkey;
```

5.1.2 Query 2

```
SELECT avg(lineitem.l_tax)
FROM
lineitem, orders
WHERE
orders.o_orderstatus='F' AND
orders.o_orderpriority='4-NOT SPECIFIED' AND
lineitem.l_linenumber=5 AND
lineitem.l_shipmode='TRUCK' AND
lineitem.l_quantity=2 AND
orders.o_orderkey=lineitem.l_orderkey;
```

5.1.3 Query 3

```
SELECT count(*) FROM lineitem WHERE l_linenumber=1;
```

5.1.4 Query 4

```
SELECT count(*) FROM lineitem WHERE l_linenumber in (1,2) AND
l_shipmode IN ('RAIL','TRUCK');
```

5.1.5 Query 5

```
SELECT count(*) FROM lineitem WHERE l_linenumber=5 AND
l_shipmode='RAIL' AND l_quantity=18;
```

6.0 Conclusion

Bizgres 0.9 introduces on-disk bitmap index, a powerful new feature for data warehousing and decision support applications. When compared to b-tree indexes, bitmap indexes offer the following benefits:

- Faster index creation time
- Smaller size on disk
- Improved query response times for standard data warehousing queries

These benefits are most prevalent when bitmap indexes are used to index columns with low cardinality data.

Appendix A: Table Definitions

LINEITEM

Column	Type	Modifiers
l_orderkey	integer	not null
l_partkey	integer	not null
l_suppkey	integer	not null
l_linenumber	integer	not null
l_quantity	numeric(15,2)	not null
l_extendedprice	numeric(15,2)	not null
l_discount	numeric(15,2)	not null
l_tax	numeric(15,2)	not null
l_returnflag	character(1)	not null
l_linestatus	character(1)	not null
l_shipdate	date	not null
l_commitdate	date	not null
l_receiptdate	date	not null
l_shipinstruct	character(25)	not null
l_shipmode	character(10)	not null
l_comment	character varying(44)	not null

ORDERS

Column	Type	Modifiers
o_orderkey	integer	not null
o_custkey	integer	not null
o_orderstatus	character(1)	not null
o_totalprice	numeric(15,2)	not null
o_orderdate	date	not null
o_orderpriority	character(15)	not null
o_clerk	character(15)	not null
o_shippriority	integer	not null

CUSTOMER

Column	Type	Modifiers
c_custkey	integer	not null
c_name	character varying(25)	not null
c_address	character varying(40)	not null
c_nationkey	integer	not null
c_phone	character(15)	not null
c_acctbal	numeric(15,2)	not null
c_mktsegment	character(10)	not null
c_comment	character varying(117)	not null
o_comment	character varying(79)	not null

Appendix B: Query Plans

QUERY 1 – BITMAP

QUERY PLAN

```
Aggregate (cost=330395.75..330395.76 rows=1 width=9)
  -> Hash Join (cost=263533.43..330395.75 rows=1 width=9)
      Hash Cond: ("outer".l_orderkey = "inner".o_orderkey)
      -> Bitmap Heap Scan on lineitem (cost=58745.35..125516.83 rows=18166
width=13)
          Recheck Cond: ((l_quantity = 5::numeric) AND (l_linenum = 5)
AND (l_shipmode = 'AIR'::bpchar))
          -> BitmapAnd (cost=58745.35..58745.35 rows=18166 width=0)
              -> Bitmap Index Scan on l_quantity_bm_idx(on-disk bitmap
index) (cost=0.00..4500.02 rows=1199721 width=0)
                  Index Cond: (l_quantity = 5::numeric)
              -> Bitmap Index Scan on l_linenum_bm_idx(on-disk bitmap
index) (cost=0.00..22779.89 rows=6278540 width=0)
                  Index Cond: (l_linenum = 5)
              -> Bitmap Index Scan on l_shipmode_bm_idx(on-disk bitmap
index) (cost=0.00..31464.94 rows=8677982 width=0)
          )
          Index Cond: (l_shipmode = 'AIR'::bpchar)
      -> Hash (cost=204786.75..204786.75 rows=533 width=4)
          -> Hash Join (cost=39338.43..204786.75 rows=533 width=4)
              Hash Cond: ("outer".o_custkey = "inner".c_custkey)
              -> Bitmap Heap Scan on orders (cost=11650.73..176756.03
rows=67538 width=8)
                  Recheck Cond: ((o_orderstatus = 'P'::bpchar) AND
(o_orderpriority = '2-HIGH'::bpchar))
                  -> BitmapAnd (cost=11650.73..11650.73 rows=67538
width=0)
                      -> Bitmap Index Scan on
o_orderstatus_bm_idx(on-disk bitmap index) (cost=0.00..1242.78 rows=3
49938 width=0)
                          Index Cond: (o_orderstatus = 'P'::bpchar)
                      -> Bitmap Index Scan on
o_orderpriority_bm_idx(on-disk bitmap index) (cost=0.00..10407.70 row
```

```

s=2894485 width=0)
                                Index Cond: (o_orderpriority = '2-
HIGH'::bpchar)
                                -> Hash   (cost=27658.10..27658.10 rows=11838 width=4)
                                -> Nested Loop (cost=1293.00..27658.10 rows=11838
width=4)
                                -> Seq Scan on nation (cost=0.00..1.31 rows=1
width=4)
                                Filter: (n_name = 'UNITED
STATES'::bpchar)
                                -> Bitmap Heap Scan on customer
(cost=1293.00..27508.81 rows=11838 width=8)
                                Recheck Cond: (("outer".n_nationkey =
customer.c_nationkey) AND (customer.c_mktsegment =
'AUTOMOBILE'::bpchar))
                                -> BitmapAnd (cost=1293.00..1293.00
rows=11838 width=0)
                                -> Bitmap Index Scan on
c_nationkey_bm_idx(on-disk bitmap index) (cost=0.00..224.
96 rows=59988 width=0)
                                Index Cond:
("outer".n_nationkey = customer.c_nationkey)
                                -> Bitmap Index Scan on
c_mktsegment_bm_idx(on-disk bitmap index) (cost=0.00..106
7.79 rows=295940 width=0)
                                Index Cond: (c_mktsegment =
'AUTOMOBILE'::bpchar)
(33 rows)
    
```

QUERY 1 – B-TREE

QUERY PLAN

```

-----
Aggregate (cost=416358.06..416358.07 rows=1 width=9)
  -> Hash Join (cost=352382.36..416358.06 rows=1 width=9)
        Hash Cond: ("outer".l_orderkey = "inner".o_orderkey)
        -> Bitmap Heap Scan on lineitem (cost=110711.06..174600.04 rows=17343
width=13)
                Recheck Cond: ((l_quantity = 5::numeric) AND (l_linenumber = 5)
AND (l_shipmode = 'AIR'::bpchar))
                -> BitmapAnd (cost=110711.06..110711.06 rows=17343 width=0)
    
```

```

        -> Bitmap Index Scan on l_quantity_btree_idx
(cost=0.00..8666.79 rows=1178797 width=0)
            Index Cond: (l_quantity = 5::numeric)
        -> Bitmap Index Scan on l_linenum_btree_idx
(cost=0.00..37817.35 rows=6058101 width=0)
            Index Cond: (l_linenum = 5)
        -> Bitmap Index Scan on l_shipmode_btree_idx
(cost=0.00..64226.42 rows=8737262 width=0)
            Index Cond: (l_shipmode = 'AIR'::bpchar)
-> Hash (cost=241670.11..241670.11 rows=473 width=4)
    -> Hash Join (cost=55124.84..241670.11 rows=473 width=4)
        Hash Cond: ("outer".o_custkey = "inner".c_custkey)
    -> Bitmap Heap Scan on orders (cost=26799.72..212935.89
rows=80874 width=8)
        Recheck Cond: ((o_orderstatus = 'P'::bpchar) AND
(o_orderpriority = '2-HIGH'::bpchar))
    -> BitmapAnd (cost=26799.72..26799.72 rows=80874
width=0)
        -> Bitmap Index Scan on
o_orderstatus_btree_idx (cost=0.00..2654.17 rows=424906 width=0)
            Index Cond: (o_orderstatus = 'P'::bpchar)
        -> Bitmap Index Scan on
o_orderpriority_btree_idx (cost=0.00..24145.30 rows=2854371 width=0)
            Index Cond: (o_orderpriority = '2-
HIGH'::bpchar)
    -> Hash (cost=28296.51..28296.51 rows=11446 width=4)
        -> Nested Loop (cost=2480.89..28296.51 rows=11446
width=4)
            -> Seq Scan on nation (cost=0.00..1.31 rows=1
width=4)
                Filter: (n_name = 'UNITED
STATES'::bpchar)
            -> Bitmap Heap Scan on customer
(cost=2480.89..28152.12 rows=11446 width=8)
                Recheck Cond: (("outer".n_nationkey =
customer.c_nationkey) AND (customer.c_mktsegment = 'AUTOMOBILE'::bpchar))
            -> BitmapAnd (cost=2480.89..2480.89
rows=11446 width=0)
                -> Bitmap Index Scan on
c_nationkey_btree_idx (cost=0.00..376.11 rows=60032 width=0)
                    Index Cond:
("outer".n_nationkey = customer.c_nationkey)
    
```

```

                                -> Bitmap Index Scan on
c_mktsegment_btree_idx (cost=0.00..2104.53 rows=286151 width=0)
                                Index Cond: (c_mktsegment =
'AUTOMOBILE'::bpchar)
(33 rows)
    
```

QUERY 2 – BITMAP

QUERY PLAN

```

-----
Aggregate (cost=713682.71..713682.72 rows=1 width=9)
  -> Merge Join (cost=706473.11..713678.58 rows=1649 width=9)
      Merge Cond: ("outer".l_orderkey = "inner".o_orderkey)
      -> Sort (cost=122798.17..122841.70 rows=17412 width=13)
          Sort Key: lineitem.l_orderkey
          -> Bitmap Heap Scan on lineitem (cost=57440.64..121571.69
rows=17412 width=13)
              Recheck Cond: ((l_quantity = 2::numeric) AND (l_linenumber
= 5) AND (l_shipmode = 'TRUCK'::bpchar))
              -> BitmapAnd (cost=57440.64..57440.64 rows=17412 width=0)
                  -> Bitmap Index Scan on l_quantity_bm_idx(on-disk
bitmap index) (cost=0.00..4500.02 rows=1199721 width=0)
                      Index Cond: (l_quantity = 2::numeric)
                  -> Bitmap Index Scan on l_linenumber_bm_idx(on-disk
bitmap index) (cost=0.00..22779.89 rows=6278540 width=0)
                      Index Cond: (l_linenumber = 5)
                  -> Bitmap Index Scan on l_shipmode_bm_idx(on-disk
bitmap index) (cost=0.00..30160.23 rows=8318065 width=0)
                      Index Cond: (l_shipmode = 'TRUCK'::bpchar)
              -> Sort (cost=583674.94..587225.94 rows=1420397 width=4)
                  Sort Key: orders.o_orderkey
                  -> Bitmap Heap Scan on orders (cost=10318.21..372165.55
rows=1420397 width=4)
                      Recheck Cond: (o_orderpriority = '4-NOT SPECIFIED'::bpchar)
                      Filter: (o_orderstatus = 'F'::bpchar)
                      -> Bitmap Index Scan on o_orderpriority_bm_idx(on-disk
bitmap index) (cost=0.00..10318.21 rows=2869489 width=0)
                          Index Cond: (o_orderpriority = '4-NOT
SPECIFIED'::bpchar)
(21 rows)
    
```

QUERY 2 – B-TREE

QUERY PLAN

```
-----  
Aggregate (cost=784550.05..784550.06 rows=1 width=9)  
  -> Merge Join (cost=777084.24..784546.08 rows=1587 width=9)  
      Merge Cond: ("outer".l_orderkey = "inner".o_orderkey)  
      -> Sort (cost=167130.59..167170.97 rows=16152 width=13)  
          Sort Key: lineitem.l_orderkey  
          -> Bitmap Heap Scan on lineitem (cost=106302.72..166001.61  
rows=16152 width=13)  
              Recheck Cond: ((l_quantity = 2::numeric) AND (l_linenum  
= 5) AND (l_shipmode = 'TRUCK'::bpchar))  
              -> BitmapAnd (cost=106302.72..106302.72 rows=16152  
width=0)  
                  -> Bitmap Index Scan on l_quantity_btree_idx  
(cost=0.00..8666.79 rows=1178797 width=0)  
                      Index Cond: (l_quantity = 2::numeric)  
                  -> Bitmap Index Scan on l_linenum_btree_idx  
(cost=0.00..37817.35 rows=6058101 width=0)  
                      Index Cond: (l_linenum = 5)  
                  -> Bitmap Index Scan on l_shipmode_btree_idx  
(cost=0.00..59818.07 rows=8137450 width=0)  
                      Index Cond: (l_shipmode = 'TRUCK'::bpchar)  
          -> Sort (cost=609953.65..613636.31 rows=1473062 width=4)  
              Sort Key: orders.o_orderkey  
              -> Bitmap Heap Scan on orders (cost=25710.65..390105.60  
rows=1473062 width=4)  
                  Recheck Cond: (o_orderpriority = '4-NOT SPECIFIED'::bpchar)  
                  Filter: (o_orderstatus = 'F'::bpchar)  
                  -> Bitmap Index Scan on o_orderpriority_btree_idx  
(cost=0.00..25710.65 rows=3039330 width=0)  
                      Index Cond: (o_orderpriority = '4-NOT  
SPECIFIED'::bpchar)  
(21 rows)
```

QUERY 3 – BITMAP

QUERY PLAN

```
-----  
Aggregate (cost=1860488.07..1860488.08 rows=1 width=0)  
  -> Bitmap Heap Scan on lineitem (cost=54770.71..1822746.84 rows=15096490  
width=0)  
    Recheck Cond: (l_linenum = 1)  
      -> Bitmap Index Scan on l_linenum_bm_idx (on-disk bitmap index)  
(cost=0.00..54770.71 rows=15096490 width=0)  
        Index Cond: (l_linenum = 1)  
  
(5 rows)
```

QUERY 3 – B-TREE

QUERY PLAN

```
-----  
Aggregate (cost=1900352.37..1900352.38 rows=1 width=0)  
  -> Bitmap Heap Scan on lineitem (cost=94353.42..1862564.21 rows=15115263  
width=0)  
    Recheck Cond: (l_linenum = 1)  
      -> Bitmap Index Scan on l_linenum_btree_idx (cost=0.00..94353.42  
rows=15115263 width=0)  
        Index Cond: (l_linenum = 1)  
  
(5 rows)
```

QUERY 4 – BITMAP

QUERY PLAN

```
-----  
Aggregate (cost=1915022.55..1915022.56 rows=1 width=0)  
  -> Bitmap Heap Scan on lineitem (cost=162679.75..1898898.59 rows=6449585  
width=0)  
    Recheck Cond: (((l_shipmode = 'RAIL'::bpchar) OR (l_shipmode =  
'TRUCK'::bpchar)) AND ((l_linenum = 1) OR (l_linenum = 2)))  
      -> BitmapAnd (cost=162679.75..162679.75 rows=7847442 width=0)  
        -> BitmapOr (cost=60755.36..60755.36 rows=16756103 width=0)  
          -> Bitmap Index Scan on l_shipmode_bm_idx (on-disk bitmap  
index) (cost=0.00..30595.13 rows=8438038 width=0)  
            Index Cond: (l_shipmode = 'RAIL'::bpchar)
```

```

        -> Bitmap Index Scan on l_shipmode_bm_idx(on-disk bitmap
index) (cost=0.00..30160.23 rows=8318065 width=0)
            Index Cond: (l_shipmode = 'TRUCK'::bpchar)
        -> BitmapOr (cost=101924.14..101924.14 rows=28093468 width=0)
            -> Bitmap Index Scan on l_linenummer_bm_idx(on-disk bitmap
index) (cost=0.00..54770.71 rows=15096490 width=0)
                Index Cond: (l_linenummer = 1)
            -> Bitmap Index Scan on l_linenummer_bm_idx(on-disk bitmap
index) (cost=0.00..47153.42 rows=12996978 width=0)
                Index Cond: (l_linenummer = 2)

(14 rows)
    
```

QUERY 4 – B-TREE

QUERY PLAN

```

-----
Aggregate (cost=2058343.57..2058343.58 rows=1 width=0)
  -> Bitmap Heap Scan on lineitem (cost=301624.59..2041836.15 rows=6602967
width=0)
    Recheck Cond: (((l_shipmode = 'RAIL'::bpchar) OR (l_shipmode =
'TRUCK'::bpchar)) AND ((l_linenummer = 1) OR (l_linenummer = 2)))
    -> BitmapAnd (cost=301624.59..301624.59 rows=8047078 width=0)
      -> BitmapOr (cost=126396.14..126396.14 rows=17194610 width=0)
        -> Bitmap Index Scan on l_shipmode_btree_idx
(cost=0.00..66578.06 rows=9057161 width=0)
          Index Cond: (l_shipmode = 'RAIL'::bpchar)
        -> Bitmap Index Scan on l_shipmode_btree_idx
(cost=0.00..59818.07 rows=8137450 width=0)
          Index Cond: (l_shipmode = 'TRUCK'::bpchar)
      -> BitmapOr (cost=175228.21..175228.21 rows=28071202 width=0)
        -> Bitmap Index Scan on l_linenummer_btree_idx
(cost=0.00..94353.42 rows=15115263 width=0)
          Index Cond: (l_linenummer = 1)
        -> Bitmap Index Scan on l_linenummer_btree_idx
(cost=0.00..80874.79 rows=12955940 width=0)
          Index Cond: (l_linenummer = 2)

(14 rows)
    
```

QUERY 5 – BITMAP

QUERY PLAN

```
Aggregate (cost=122934.20..122934.21 rows=1 width=0)
  -> Bitmap Heap Scan on lineitem (cost=57875.55..122890.04 rows=17664
width=0)
    Recheck Cond: ((l_quantity = 18::numeric) AND (l_linenum = 5) AND
(l_shipmode = 'RAIL'::bpchar))
      -> BitmapAnd (cost=57875.55..57875.55 rows=17664 width=0)
        -> Bitmap Index Scan on l_quantity_bm_idx(on-disk bitmap index)
(cost=0.00..4500.02 rows=1199721 width=0)
          Index Cond: (l_quantity = 18::numeric)
        -> Bitmap Index Scan on l_linenum_bm_idx(on-disk bitmap
index) (cost=0.00..22779.89 rows=6278540 width=0)
          Index Cond: (l_linenum = 5)
        -> Bitmap Index Scan on l_shipmode_bm_idx(on-disk bitmap index)
(cost=0.00..30595.13 rows=8438038 width=0)
          Index Cond: (l_shipmode = 'RAIL'::bpchar)
(10 rows)
```

QUERY 5 – B-TREE

QUERY PLAN

```
Aggregate (cost=179221.59..179221.60 rows=1 width=0)
  -> Bitmap Heap Scan on lineitem (cost=113062.71..179176.64 rows=17978
width=0)
    Recheck Cond: ((l_quantity = 18::numeric) AND (l_linenum = 5) AND
(l_shipmode = 'RAIL'::bpchar))
      -> BitmapAnd (cost=113062.71..113062.71 rows=17978 width=0)
        -> Bitmap Index Scan on l_quantity_btree_idx
(cost=0.00..8666.79 rows=1178797 width=0)
          Index Cond: (l_quantity = 18::numeric)
        -> Bitmap Index Scan on l_linenum_btree_idx
(cost=0.00..37817.35 rows=6058101 width=0)
          Index Cond: (l_linenum = 5)
        -> Bitmap Index Scan on l_shipmode_btree_idx
(cost=0.00..66578.06 rows=9057161 width=0)
```

Index Cond: (l_shipmode = 'RAIL'::bpchar)

(10 rows)