付録資料「DBT-3 トランザクション特性の解析」

1 はじめに

本資料では、PostgreSQL7.4.6 および 8.0.0beta5 を OSDL-DBT-3 で実行した結果を解析し、よりよいパフォーマンスを得るためにチューニング作業を行った結果を報告する。

なお、本資料では PostgreSQL7.4.6 を「7.4」、PostgreSQL8.0.0beta5 を「8.0」と表記する。

2 個々の問い合わせごとの解析とチューニング

本節では、OSDL-DBT-3 の Q1 から Q22 までの 22 個の問い合わせを順に解析し、チューニングした結果を報告する。 なお、問い合わせによってはチューニングの余地のないもののあることをあらかじめお断りしておく。

また、解析の対象としたのはスケーリングファクタ 1 のデータのみであるが、チューニング結果は異なるスケーリングファクタにも適用できるはずである。

2.1 Q1

問い合わせをリスト 2.1-1 に示す。 なお、一部のパラメータは実行毎にランダムに選ばれるので、ここであげた問い合わせはあくまで一例として考えていただきたい。 ただし、その場合でも本節で解説したチューニングテクニックは適用できるはずである (他の問い合わせに関しても以下同様)。

リスト 2.1-1　Q1 の問い合わせ

```
select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty,sum(l_extendedprice)
as sum_base_price, sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge, avg(l_quantity)
as avg_qty, avg(l_extendedprice) as avg_price, avg(l_discount) as avg_disc, count(*)
as count_order from lineitem where l_shipdate <= date'1998-12-01' - interval '108
days' group by l_returnflag, l_linestatus order by l_returnflag, l_linestatus;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.1-2 に示す。EXPLAIN ANALYZE コマンドでは、実際に問い合わせが実行されるので、PostgreSQL が見積もった行数、コストと、実際の行数、実行時間との比較ができる。

リスト 2.1-2　Q1 の実行計画

```
Sort  (cost=267537.86..267537.87 rows=3 width=26) (actual
time=48822.346..48822.348 rows=4 loops=1)
  Sort Key: l_returnflag, l_linestatus
  ->  HashAggregate  (cost=267537.73..267537.83 rows=3 width=26) (actual
time=48822.317..48822.329 rows=4 loops=1)
        ->  Seq Scan on lineitem  (cost=0.00..217568.53 rows=1998768 width=26)
(actual time=11.158..11895.998 rows=5879439 loops=1)
 Filter: ((l_shipdate)::timestamp without time zone <= '1998-08-15
00:00:00'::timestamp without time zone)
 Total runtime: 48822.474 ms
(6 rows)
```

・ 実行計画の理解

この実行計画を下から読むことによって理解していく。

まず、「Seq Scan on lineitem」によって、lineitem テーブルを順スキャン(sequencial scan)すなわち、インデックスなど使用せずにテーブルファイルをそのまま読み込むことによって処理をしていく．このときのコストの見積もりは「(cost=0.00..217568.53 rows=1998768 width=26)」から、217568.53 である。 この数値の単位は、ディスクから PostgreSQL がデータを 1 ページ(通常 8192 バイト)読み込むのに要するコストであるが、現実にかかった時間とは直接関係はない。 実際に要した時間は「(actual time=11.158..11895.998rows=5879439 loops=1)」から、11895.998-11.158 = 11884.84 ミリ秒であることがわかる。

lineitem から読み出したデータは、「HashAggregate(cost=267537.73..267537.83 rows=3 width=26) (actualtime=48822.317..48822.329 rows=4 loops=1)」によって、「group by l_returnflag, l_linestatus」の部分が処理される。 HashAggregate は、ハッシュ表を使って効率的に group by 処理に必要な行の分類を行うことができる。ただし、ハッシュ表を作るまでに 48822.317 ミリ秒かかっている。 実際にハッシュ表を作るために要した時間は、下位ノードの実行時間を引いて、48822.317 - 11895.998 = 36926.319 ミリ秒であることがわかる。

最後は order by l_returnflag, l_linestatus の処理である。 実行計画では、

```
Sort (cost=267537.86..267537.87 rows=3 width=26)(actual time=48822.346..48822.348
rows=4 loops=1)
```

に対応する。 一般にソート処理はコストが高いが、ここでは「rows=4」から、4 行しかソート対象がないこともあり、ほとんど処理に時間がかかっていない。

・ チューニング

この実行計画の中で時間がかかっているのは、ハッシュ表の作成処理と lineitem テーブルへの順スキャンである。　このうち、ハッシュ表の作成処理は改善の余地がないため、ここでは lineitem テーブルの処理時間を短縮することを考える。

lineitem テーブルをアクセスするのに 11 秒以上時間がかかっているが、これは、lineitem テーブルが 600 万行以上、サイズにして 1G バイト以上あり、順スキャンではテーブルサイズに比例したアクセス時間がかかるためである。

もしも WHERE 句によって読み込みに必要な行を充分制限できることが分かっていれば、インデックスを使ってディスクからの読み込みデータ量を減らすことができるが、今回は 5879439 行、すなわち 600 万行のうち大半の行をアクセスするため、インデックスを経由したアクセスではかえって遅くなってしまう。

結論として、今回のケースではチューニングの余地はないことになる。

では、検索範囲が狭くなればインデックスが使われるだろうか？

今回の問い合わせでは、検索条件として以下が指定されている。

```
where l_shipdate <= date'1998-12-01' - interval '108 days'
```

実行計画の対応部分は以下になる。

```
Filter: ((l_shipdate)::timestamp without time zone <= '1998-08-15
00:00:00'::timestamp without time zone)
```

7.4 では、このままでは l_shipdate に設定したインデックスが使われない。

l_shipdae は date 型であり、date'1998-12-01' - interval '108 days'を計算した結果のデータ型である timestamp without time zone と一致しないからである。インデックスを使わせるためには、date'1998-12-01' - interval '108days'を計算した結果を date 型にキャスト(型変換)すればよい。実際の問い合わせはリスト 2.1-3 のようになる。

リスト 2.1-3　改善された問い合わせ

```
select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty,sum(l_extendedprice)
as sum_base_price, sum(l_extendedprice *(1 - l_discount)) as sum_disc_price,
sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge, avg(l_quantity)
as avg_qty, avg(l_extendedprice) as avg_price, avg(l_discount) as avg_disc, count(*)
as count_order from lineitem where l_shipdate <=
(date'1998-12-01' - interval '108 days')::date group by l_returnflag, l_linestatus
order by l_returnflag, l_linestatus;
```

検索条件を変えて、インデックスが使われるケースをリスト 2.1-4 に示す。

リスト 2.1-4　インデックスが使われるケース

```
explain analyze select l_returnflag, l_linestatus, sum(l_quantity) as
sum_qty,sum(l_extendedprice) as sum_base_price, sum(l_extendedprice * (1 -
l_discount)) as sum_disc_price, sum(l_extendedprice * (1 - l_discount) * (1 + l_tax))
as sum_charge, avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price,
avg(l_discount) as avg_disc, count(*) as count_order from lineitem where l_shipdate
<= (date'1992-5-01' - interval '108 days')::date group by l_returnflag, l_linestatus
order by l_returnflag, l_linestatus;

------------------------------------------------------------------------------
---------------------------------------
                                                QUERY PLAN
------------------------------------------------------------------------------
---------------------------------------
 Sort  (cost=2379.12..2379.12 rows=1 width=26) (actual time=61.432..61.433 rows=2
loops=1)
   Sort Key: l_returnflag, l_linestatus
   ->  HashAggregate  (cost=2379.07..2379.11 rows=1 width=26) (actual
time=61.411..61.417 rows=2 loops=1)
         ->  Index Scan using i_l_shipdate on lineitem  (cost=0.00..2364.32
rows=590 width=26) (actual time=0.025..46.986 rows=1900 loops=1)
               Index Cond: (l_shipdate <= '1992-01-14'::date)
 Total runtime: 61.563 ms
(6 rows)
```

このように、検索範囲が充分狭ければわずか 62 ミリ秒ほどで検索が実行できる。

なお、PostgreSQL 8.0 ではオプティマイザが型の違いを認識するので、問い合わせを書き換える必要はない。

2.2 Q2

問い合わせをリスト 2.2-1 に示す。

リスト 2.2-1　Q2 の問い合わせ

```
select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
```

```
from part, supplier, partsupp, nation, region where p_partkey = ps_partkey and
s_suppkey = ps_suppkey and p_size = 41 and p_type like '%STEEL' and s_nationkey =
n_nationkey and n_regionkey = r_regionkey and r_name = 'AMERICA' and ps_supplycost
= ( select min(ps_supplycost) from partsupp, supplier, nation, region where p_partkey
= ps_partkey and s_suppkey = ps_suppkey and s_nationkey = n_nationkey and n_regionkey
= r_regionkey and r_name = 'AMERICA' ) order by s_acctbal desc, n_name, s_name,
p_partkey LIMIT 100;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.2-2 に示す。

リスト 2.2-2　Q2 の実行計画

```
Limit  (cost=7945.15..7945.16 rows=1 width=208) (actual time=751.725..751.910
rows=100 loops=1)
   ->  Sort  (cost=7945.15..7945.16 rows=1 width=208) (actual
time=751.720..751.782 rows=100 loops=1)
         Sort Key: supplier.s_acctbal, nation.n_name, supplier.s_name,
part.p_partkey
         ->  Hash Join  (cost=7943.73..7945.14 rows=1 width=208) (actual
time=746.161..750.199 rows=508 loops=1)
               Hash Cond: ("outer".n_regionkey = "inner".r_regionkey)
               ->  Hash Join  (cost=7942.66..7944.06 rows=2 width=212) (actual
time=746.080..748.086 rows=508 loops=1)
                     Hash Cond: ("outer".n_nationkey = "inner".s_nationkey)
                     ->  Seq Scan on nation  (cost=0.00..1.25 rows=25 width=37)
(actual time=0.003..0.031 rows=25 loops=1)
                     ->  Hash  (cost=7942.66..7942.66 rows=1 width=183) (actual
time=746.055..746.055 rows=0 loops=1)
                           ->  Nested Loop  (cost=0.00..7942.66 rows=1 width=183)
(actual time=1.186..744.582 rows=508 loops=1)
                                 ->  Nested Loop  (cost=0.00..7939.63 rows=1
width=37) (actual time=1.168..736.561 rows=508 loops=1)
                                       Join Filter: ("inner".ps_supplycost =
(subplan))
                                       ->  Seq Scan on part  (cost=0.00..7757.93
rows=2 width=33) (actual time=0.114..200.299 rows=800 loops=1)
                                             Filter: ((p_size = 41) AND
((p_type)::text ~~ '%STEEL'::text))
```

```
                                        ->  Index Scan using i_ps_partkey on
partsupp  (cost=0.00..3.06 rows=4 width=12) (actual time=0.060..0.068 rows=4
loops=800)
                                              Index Cond: ("outer".p_partkey =
partsupp.ps_partkey)
                                      SubPlan
                                        ->  Aggregate  (cost=20.78..20.78 rows=1
width=4) (actual time=0.145..0.145 rows=1 loops=3200)
                                              ->  Hash Join  (cost=19.27..20.77
rows=3 width=4) (actual time=0.118..0.141 rows=1 loops=3200)
                                                    Hash Cond:
("outer".n_regionkey = "inner".r_regionkey)
                                                    ->  Hash Join
(cost=18.20..19.65 rows=6 width=8) (actual time=0.094..0.133 rows=4 loops=3200)
                                                          Hash Cond:
("outer".n_nationkey = "inner".s_nationkey)
                                                          ->  Seq Scan on nation
(cost=0.00..1.25 rows=25 width=8) (actual time=0.002..0.024 rows=25 loops=3200)
                                                          ->  Hash
(cost=18.19..18.19 rows=6 width=8) (actual time=0.076..0.076 rows=0 loops=3200)
                                                                ->  Nested Loop
(cost=0.00..18.19 rows=6 width=8) (actual time=0.020..0.070 rows=4 loops=3200)
                                                                      ->  Index
Scan using i_ps_partkey on partsupp  (cost=0.00..3.06 rows=5 width=8) (actual
time=0.007..0.015 rows=4 loops=3200)

Index Cond: ($0 = ps_partkey)
                                                                      ->  Index
Scan using supplier_pkey on supplier  (cost=0.00..3.01 rows=1 width=8) (actual
time=0.007..0.009 rows=1 loops=12800)

Index Cond: (supplier.s_suppkey = "outer".ps_suppkey)
                                                    ->  Hash  (cost=1.06..1.06
rows=2 width=4) (actual time=0.017..0.017 rows=0 loops=1)
                                                          ->  Seq Scan on region
(cost=0.00..1.06 rows=2 width=4) (actual time=0.010..0.013 rows=1 loops=1)
                                                                Filter: (r_name
= 'AMERICA'::bpchar)
                                  ->  Index Scan using supplier_pkey on supplier
(cost=0.00..3.01 rows=1 width=154) (actual time=0.006..0.007 rows=1 loops=508)
```

```
                                    Index Cond: (supplier.s_suppkey =
"outer".ps_suppkey)
               ->  Hash  (cost=1.06..1.06 rows=2 width=4) (actual
time=0.056..0.056 rows=0 loops=1)
                     ->  Seq Scan on region  (cost=0.00..1.06 rows=2 width=4)
(actual time=0.047..0.051 rows=1 loops=1)
                           Filter: (r_name = 'AMERICA'::bpchar)
 Total runtime: 752.559 ms
(38 rows)
```

part テーブルから p_size = 41 and p_type like '%STEEL'という条件で検索する部分がボトルネックになっている。「p_type like '%STEEL '」の部分は後方一致になっているため、インデックスが使用できないが、まず p_size = 41 をインデックス検索し、その結果から「p_type like '%STEEL'」を得ることによって高速化できる。そこで、次のようにインデックスを設定する。

```
create index i_p_size on part(p_size);
```

また、このままではインデックスが使用されないので，postgresql.conf の以下の項目を設定してオプティマイザがインデックスを使用する傾向を強める。

```
set random_page_cost = 2;
```

なお、random_page_cost は、インデックス経由でデータをアクセスするときのコストで、数値の単位は順スキャンでデータをアクセスするときのコストを 1 としたものである。random_page_cost のデフォルト値は 4 である。

変更後の実行計画をリスト 2.2-3 に示す。

リスト 2.2-3　改善された問い合わせ

```
Limit  (cost=6838.42..6838.43 rows=1 width=208) (actual time=595.478..595.666
rows=100 loops=1)
   ->  Sort  (cost=6838.42..6838.43 rows=1 width=208) (actual
time=595.474..595.540 rows=100 loops=1)
         Sort Key: supplier.s_acctbal, nation.n_name, supplier.s_name,
part.p_partkey
         ->  Hash Join  (cost=6837.00..6838.41 rows=1 width=208) (actual
time=590.432..594.410 rows=508 loops=1)
               Hash Cond: ("outer".n_regionkey = "inner".r_regionkey)
```

```
              ->  Hash Join  (cost=6835.93..6837.32 rows=2 width=212) (actual
time=590.333..592.336 rows=508 loops=1)
                    Hash Cond: ("outer".n_nationkey = "inner".s_nationkey)
                    ->  Seq Scan on nation  (cost=0.00..1.25 rows=25 width=37)
(actual time=0.003..0.031 rows=25 loops=1)
                    ->  Hash  (cost=6835.93..6835.93 rows=1 width=183) (actual
time=590.308..590.308 rows=0 loops=1)
                          ->  Nested Loop  (cost=0.00..6835.93 rows=1 width=183)
(actual time=0.858..588.724 rows=508 loops=1)
                                ->  Nested Loop  (cost=0.00..6832.90 rows=1
width=37) (actual time=0.845..581.078 rows=508 loops=1)
                                      Join Filter: ("inner".ps_supplycost =
(subplan))
                                      ->  Index Scan using i_p_size on part
(cost=0.00..6582.36 rows=2 width=33) (actual time=0.118..62.305 rows=800 loops=1)
                                            Index Cond: (p_size = 41)
                                            Filter: ((p_type)::text ~~
'%STEEL'::text)
                                      ->  Index Scan using i_ps_partkey on
partsupp  (cost=0.00..3.07 rows=5 width=12) (actual time=0.040..0.049 rows=4
loops=800)
                                            Index Cond: ("outer".p_partkey =
partsupp.ps_partkey)
                                      SubPlan
                                        ->  Aggregate  (cost=23.84..23.84 rows=1
width=4) (actual time=0.144..0.145 rows=1 loops=3200)
                                              ->  Hash Join  (cost=22.31..23.83
rows=3 width=4) (actual time=0.118..0.140 rows=1 loops=3200)
                                                    Hash Cond:
("outer".n_regionkey = "inner".r_regionkey)
                                                    ->  Hash Join
(cost=21.24..22.70 rows=7 width=8) (actual time=0.093..0.132 rows=4 loops=3200)
                                                          Hash Cond:
("outer".n_nationkey = "inner".s_nationkey)
                                                          ->  Seq Scan on nation
(cost=0.00..1.25 rows=25 width=8) (actual time=0.002..0.025 rows=25 loops=3200)
                                                          ->  Hash
(cost=21.23..21.23 rows=7 width=8) (actual time=0.076..0.076 rows=0 loops=3200)
                                                                ->  Nested Loop
(cost=0.00..21.23 rows=7 width=8) (actual time=0.020..0.070 rows=4 loops=3200)
```

```
                                                                    ->  Index
Scan using i_ps_partkey on partsupp  (cost=0.00..3.07 rows=6 width=8) (actual
time=0.008..0.016 rows=4 loops=3200)

Index Cond: ($0 = ps_partkey)
                                                                    ->  Index
Scan using supplier_pkey on supplier  (cost=0.00..3.01 rows=1 width=8) (actual
time=0.007..0.009 rows=1 loops=12800)

Index Cond: (supplier.s_suppkey = "outer".ps_suppkey)
                                                        ->  Hash  (cost=1.06..1.06
rows=2 width=4) (actual time=0.014..0.014 rows=0 loops=1)
                                                              ->  Seq Scan on region
(cost=0.00..1.06 rows=2 width=4) (actual time=0.007..0.010 rows=1 loops=1)
                                                                    Filter: (r_name
= 'AMERICA'::bpchar)
                                  ->  Index Scan using supplier_pkey on supplier
(cost=0.00..3.01 rows=1 width=154) (actual time=0.006..0.007 rows=1 loops=508)
                                        Index Cond: (supplier.s_suppkey =
"outer".ps_suppkey)
               ->  Hash  (cost=1.06..1.06 rows=2 width=4) (actual
time=0.078..0.078 rows=0 loops=1)
                     ->  Seq Scan on region  (cost=0.00..1.06 rows=2 width=4)
(actual time=0.070..0.075 rows=1 loops=1)
                           Filter: (r_name = 'AMERICA'::bpchar)
 Total runtime: 596.065 ms
(39 rows)
```

このように、実行時間が 752.559 ms から 596.065 ms に短縮された。

PostgreSQL 8.0 では多少実行計画が変わり(一番外側のループが Hash Join から Nested Loop)、7.4 に比べて多少実行時間が短縮されている(リスト 2.2-4)。

リスト 2.2-4　PostgreSQL8.0 での実行計画

```
Limit  (cost=10794.09..10794.09 rows=1 width=209) (actual time=546.159..546.341
rows=100 loops=1)
   ->  Sort  (cost=10794.09..10794.09 rows=1 width=209) (actual
time=546.155..546.216 rows=100 loops=1)
         Sort Key: supplier.s_acctbal, nation.n_name, supplier.s_name,
```

```
part.p_partkey
         ->  Nested Loop  (cost=1.06..10794.08 rows=1 width=209) (actual
time=0.803..544.727 rows=508 loops=1)
               Join Filter: ("outer".s_nationkey = "inner".n_nationkey)
               ->  Nested Loop  (cost=0.00..10791.53 rows=1 width=184) (actual
time=0.757..508.448 rows=508 loops=1)
                     ->  Nested Loop  (cost=0.00..10788.50 rows=1 width=37)
(actual time=0.743..500.132 rows=508 loops=1)
                           Join Filter: ("inner".ps_supplycost = (subplan))
                           ->  Seq Scan on part  (cost=0.00..7863.07 rows=2
width=33) (actual time=0.162..199.928 rows=800 loops=1)
                                 Filter: ((p_size = 41) AND ((p_type)::text ~~
'%STEEL'::text))
                           ->  Index Scan using i_ps_partkey on partsupp
(cost=0.00..3.28 rows=21 width=12) (actual time=0.040..0.046 rows=4 loops=800)
                                 Index Cond: ("outer".p_partkey =
partsupp.ps_partkey)
                           SubPlan
                             ->  Aggregate  (cost=69.48..69.48 rows=1 width=4)
(actual time=0.077..0.077 rows=1 loops=3200)
                                   ->  Hash Join  (cost=2.50..69.47 rows=4
width=4) (actual time=0.051..0.073 rows=1 loops=3200)
                                         Hash Cond: ("outer".s_nationkey =
"inner".n_nationkey)
                                         ->  Nested Loop  (cost=0.00..66.83
rows=20 width=8) (actual time=0.019..0.066 rows=4 loops=3200)
                                               ->  Index Scan using i_ps_partkey
on partsupp  (cost=0.00..3.28 rows=21 width=8) (actual time=0.007..0.014 rows=4
loops=3200)
                                                     Index Cond: ($0 = ps_partkey)
                                               ->  Index Scan using supplier_pkey
on supplier  (cost=0.00..3.01 rows=1 width=8) (actual time=0.006..0.008 rows=1
loops=12800)
                                                     Index Cond:
(supplier.s_suppkey = "outer".ps_suppkey)
                                         ->  Hash  (cost=2.49..2.49 rows=5
width=4) (actual time=0.120..0.120 rows=0 loops=1)
                                               ->  Hash Join  (cost=1.06..2.49
rows=5 width=4) (actual time=0.064..0.115 rows=5 loops=1)
                                                     Hash Cond:
```

```
("outer".n_regionkey = "inner".r_regionkey)
                                                    ->  Seq Scan on nation
(cost=0.00..1.25 rows=25 width=8) (actual time=0.004..0.029 rows=25 loops=1)
                                                    ->  Hash  (cost=1.06..1.06
rows=1 width=4) (actual time=0.044..0.044 rows=0 loops=1)
                                                          ->  Seq Scan on region
(cost=0.00..1.06 rows=1 width=4) (actual time=0.037..0.040 rows=1 loops=1)
                                                                Filter: (r_name
= 'AMERICA'::bpchar)
                    ->  Index Scan using supplier_pkey on supplier
(cost=0.00..3.01 rows=1 width=155) (actual time=0.006..0.007 rows=1 loops=508)
                          Index Cond: (supplier.s_suppkey = "outer".ps_suppkey)
              ->  Hash Join  (cost=1.06..2.49 rows=5 width=33) (actual
time=0.009..0.058 rows=5 loops=508)
                    Hash Cond: ("outer".n_regionkey = "inner".r_regionkey)
                    ->  Seq Scan on nation  (cost=0.00..1.25 rows=25 width=37)
(actual time=0.003..0.027 rows=25 loops=508)
                    ->  Hash  (cost=1.06..1.06 rows=1 width=4) (actual
time=0.013..0.013 rows=0 loops=1)
                          ->  Seq Scan on region  (cost=0.00..1.06 rows=1 width=4)
(actual time=0.007..0.011 rows=1 loops=1)
                                Filter: (r_name = 'AMERICA'::bpchar)
 Total runtime: 546.766 ms
(37 rows)
```

8.0 でも、7.4 と同様にインデックスを設定するなどの処置を施すことにより、更に実行時間が短くなった(リスト 2.2-5)。

```
create index i_p_size on part(p_size);
set random_page_cost = 2;
```

リスト 2.2-5　PostgreSQL8.0 での改善された実行計画

```
Limit  (cost=9539.35..9539.35 rows=1 width=209) (actual time=400.246..400.456
rows=100 loops=1)
   ->  Sort  (cost=9539.35..9539.35 rows=1 width=209) (actual
time=400.242..400.333 rows=100 loops=1)
         Sort Key: supplier.s_acctbal, nation.n_name, supplier.s_name,
part.p_partkey
```

```
        ->  Nested Loop  (cost=1.06..9539.34 rows=1 width=209) (actual
time=0.881..398.883 rows=508 loops=1)
              Join Filter: ("outer".s_nationkey = "inner".n_nationkey)
              ->  Nested Loop  (cost=0.00..9536.78 rows=1 width=184) (actual
time=0.835..363.342 rows=508 loops=1)
                    ->  Nested Loop  (cost=0.00..9533.76 rows=1 width=37) (actual
time=0.822..355.182 rows=508 loops=1)
                          Join Filter: ("inner".ps_supplycost = (subplan))
                          ->  Index Scan using i_p_size on part
(cost=0.00..6608.33 rows=2 width=33) (actual time=0.221..57.297 rows=800 loops=1)
                                Index Cond: (p_size = 41)
                                Filter: ((p_type)::text ~~ '%STEEL'::text)
                          ->  Index Scan using i_ps_partkey on partsupp
(cost=0.00..3.28 rows=21 width=12) (actual time=0.038..0.045 rows=4 loops=800)
                                Index Cond: ("outer".p_partkey =
partsupp.ps_partkey)
                          SubPlan
                            ->  Aggregate  (cost=69.48..69.48 rows=1 width=4)
(actual time=0.076..0.077 rows=1 loops=3200)
                                  ->  Hash Join  (cost=2.50..69.47 rows=4
width=4) (actual time=0.051..0.073 rows=1 loops=3200)
                                        Hash Cond: ("outer".s_nationkey =
"inner".n_nationkey)
                                        ->  Nested Loop  (cost=0.00..66.83
rows=20 width=8) (actual time=0.019..0.066 rows=4 loops=3200)
                                              ->  Index Scan using i_ps_partkey
on partsupp  (cost=0.00..3.28 rows=21 width=8) (actual time=0.007..0.014 rows=4
loops=3200)
                                                    Index Cond: ($0 = ps_partkey)
                                              ->  Index Scan using supplier_pkey
on supplier  (cost=0.00..3.01 rows=1 width=8) (actual time=0.006..0.008 rows=1
loops=12800)
                                                    Index Cond:
(supplier.s_suppkey = "outer".ps_suppkey)
                                        ->  Hash  (cost=2.49..2.49 rows=5
width=4) (actual time=0.124..0.124 rows=0 loops=1)
                                              ->  Hash Join  (cost=1.06..2.49
rows=5 width=4) (actual time=0.066..0.115 rows=5 loops=1)
                                                    Hash Cond:
("outer".n_regionkey = "inner".r_regionkey)
```

```
                                                    ->  Seq Scan on nation
(cost=0.00..1.25 rows=25 width=8) (actual time=0.004..0.028 rows=25 loops=1)
                                                    ->  Hash  (cost=1.06..1.06
rows=1 width=4) (actual time=0.046..0.046 rows=0 loops=1)
                                                          ->  Seq Scan on region
(cost=0.00..1.06 rows=1 width=4) (actual time=0.038..0.041 rows=1 loops=1)
                                                                Filter: (r_name
= 'AMERICA'::bpchar)
                    ->  Index Scan using supplier_pkey on supplier
(cost=0.00..3.01 rows=1 width=155) (actual time=0.006..0.007 rows=1 loops=508)
                          Index Cond: (supplier.s_suppkey = "outer".ps_suppkey)
              ->  Hash Join  (cost=1.06..2.49 rows=5 width=33) (actual
time=0.009..0.057 rows=5 loops=508)
                    Hash Cond: ("outer".n_regionkey = "inner".r_regionkey)
                    ->  Seq Scan on nation  (cost=0.00..1.25 rows=25 width=37)
(actual time=0.003..0.027 rows=25 loops=508)
                    ->  Hash  (cost=1.06..1.06 rows=1 width=4) (actual
time=0.014..0.014 rows=0 loops=1)
                          ->  Seq Scan on region  (cost=0.00..1.06 rows=1 width=4)
(actual time=0.008..0.011 rows=1 loops=1)
                                Filter: (r_name = 'AMERICA'::bpchar)
 Total runtime: 400.884 ms
(38 rows)
```

2.3 Q3

問い合わせをリスト 2.3-1 に示す。

リスト 2.3-1　Q3 の問い合わせ

```
select l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue, o_orderdate,
o_shippriority from customer, orders, lineitem where c_mktsegment = 'BUILDING' and
c_custkey = o_custkey and l_orderkey = o_orderkey and o_orderdate < date '1995-03-11'
and l_shipdate > date '1995-03-11' group by l_orderkey, o_orderdate, o_shippriority
order by revenue desc, o_orderdate LIMIT 10;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.3-2 に示す。

リスト 2.3-2　Q3 の実行計画

```
Limit  (cost=379686.28..379686.30 rows=10 width=20) (actual
time=24354.779..24354.799 rows=10 loops=1)
   ->  Sort  (cost=379686.28..380329.19 rows=257165 width=20) (actual
time=24354.773..24354.780 rows=10 loops=1)
         Sort Key: sum((lineitem.l_extendedprice * (1::double precision -
lineitem.l_discount))), orders.o_orderdate
         ->  GroupAggregate  (cost=346523.70..351667.00 rows=257165 width=20)
(actual time=24187.226..24283.039 rows=11616 loops=1)
               ->  Sort  (cost=346523.70..347166.61 rows=257165 width=20) (actual
time=24187.151..24216.732 rows=30527 loops=1)
                     Sort Key: lineitem.l_orderkey, orders.o_orderdate,
orders.o_shippriority
                     ->  Merge Join  (cost=85040.97..318504.42 rows=257165
width=20) (actual time=8281.834..24030.242 rows=30527 loops=1)
                           Merge Cond: ("outer".l_orderkey = "inner".o_orderkey)
                           ->  Index Scan using i_l_orderkey on lineitem
(cost=0.00..221707.08 rows=3159610 width=12) (actual time=0.114..12130.160
rows=3251825 loops=1)
                                 Filter: (l_shipdate > '1995-03-11'::date)
                           ->  Sort  (cost=85040.97..85373.78 rows=133123
width=12) (actual time=8281.287..8476.093 rows=165577 loops=1)
                                 Sort Key: orders.o_orderkey
                                 ->  Hash Join  (cost=5997.29..72337.38
rows=133123 width=12) (actual time=233.454..7569.458 rows=146666 loops=1)
                                       Hash Cond: ("outer".o_custkey =
"inner".c_custkey)
                                       ->  Seq Scan on orders
(cost=0.00..47282.62 rows=711882 width=16) (actual time=0.639..1818.280 rows=724706
loops=1)
                                             Filter: (o_orderdate <
'1995-03-11'::date)
                                       ->  Hash  (cost=5831.12..5831.12
rows=28067 width=4) (actual time=232.695..232.695 rows=0 loops=1)
                                             ->  Seq Scan on customer
(cost=0.00..5831.12 rows=28067 width=4) (actual time=0.026..192.595 rows=30142
loops=1)
                                                   Filter: (c_mktsegment =
'BUILDING'::bpchar)
```

```
 Total run time: 24364.934 ms
(20 rows)
```

この実行計画では、(group by l_orderkey, o_orderdate, o_shippriority)の部分を処理するために比較的処理時間のかかる GroupAggregate を使用しているが、ソートメモリを増やすことによって更に効率のよい HashAggregate にすることができる。ソートメモリの設定は postgresql.conf で以下のように行う。

```
set sort_mem to 102400;
```

この例では 102400*1KB = 100MB のソートメモリを設定している。

リスト 2.3-3　改善された問い合わせ計画

```
Limit  (cost=313832.01..313832.04 rows=10 width=20) (actual
time=18594.485..18594.503 rows=10 loops=1)
   ->  Sort  (cost=313832.01..314597.29 rows=306112 width=20) (actual
time=18594.478..18594.483 rows=10 loops=1)
         Sort Key: sum((lineitem.l_extendedprice * (1::double precision -
lineitem.l_discount))), orders.o_orderdate
         ->  HashAggregate  (cost=283643.71..285939.55 rows=306112 width=20)
(actual time=18535.316..18559.915 rows=11622 loops=1)
               ->  Hash Join  (cost=58603.64..280582.59 rows=306112 width=20)
(actual time=3222.820..18424.091 rows=30550 loops=1)
                     Hash Cond: ("outer".l_orderkey = "inner".o_orderkey)
                     ->  Seq Scan on lineitem  (cost=0.00..202840.74 rows=3215416
width=12) (actual time=0.053..8010.114 rows=3255020 loops=1)
                           Filter: (l_shipdate > '1995-03-11'::date)
                     ->  Hash  (cost=58246.28..58246.28 rows=142945 width=12)
(actual time=3220.257..3220.257 rows=0 loops=1)
                           ->  Hash Join  (cost=5904.63..58246.28 rows=142945
width=12) (actual time=238.685..3039.861 rows=146721 loops=1)
                                 Hash Cond: ("outer".o_custkey =
"inner".c_custkey)
                                 ->  Seq Scan on orders  (cost=0.00..47320.75
rows=718290 width=16) (actual time=0.060..1825.915 rows=725487 loops=1)
                                       Filter: (o_orderdate < '1995-03-11'::date)
                                 ->  Hash  (cost=5830.00..5830.00 rows=29851
width=4) (actual time=237.195..237.195 rows=0 loops=1)
```

```
                                    ->  Seq Scan on customer
(cost=0.00..5830.00 rows=29851 width=4) (actual time=0.099..204.183 rows=30142
loops=1)
                                          Filter: (c_mktsegment =
'BUILDING'::bpchar)
 Total runtime: 18607.041 ms
(17 rows)
```

更なる改善案としては、o_orderdate < date '1995-03-11'や l_shipdate >date '1995-03-11'でインデックスを使用することも考えられるが、o_orderdate < date '1995-03-11'の選択率は 48%、l_shipdate >date '1995-03-11'の選択率は 54%と大きいため、インデックスの効果は期待できないと考えられる。実際、PostgreSQL のオプティマイザはインデックスを使用しないプランを選択している。

PostgreSQL 8.0 でも、7.4 と同様に sort_mem を 102400 にすることにより、改善が見られた。

リスト 2.3-4　PostgreSQL8.0 での最初の実行計画

```
Limit  (cost=427508.02..427508.04 rows=10 width=20) (actual
time=25114.838..25114.860 rows=10 loops=1)
   ->  Sort  (cost=427508.02..428333.06 rows=330017 width=20) (actual
time=25114.834..25114.842 rows=10 loops=1)
         Sort Key: sum((lineitem.l_extendedprice * (1::double precision -
lineitem.l_discount))), orders.o_orderdate
         ->  GroupAggregate  (cost=374318.02..380918.36 rows=330017 width=20)
(actual time=24954.751..25044.744 rows=11613 loops=1)
               ->  Sort  (cost=374318.02..375143.07 rows=330017 width=20) (actual
time=24954.682..24983.977 rows=30511 loops=1)
                     Sort Key: lineitem.l_orderkey, orders.o_orderdate,
orders.o_shippriority
                     ->  Merge Join  (cost=89864.84..327728.37 rows=330017
width=20) (actual time=10713.554..24805.904 rows=30511 loops=1)
                           Merge Cond: ("outer".l_orderkey = "inner".o_orderkey)
                           ->  Index Scan using i_l_orderkey on lineitem
(cost=0.00..224768.89 rows=3258404 width=12) (actual time=5.260..10782.281
rows=3251844 loops=1)
                                 Filter: (l_shipdate > '1995-03-11'::date)
                           ->  Sort  (cost=89864.84..90244.60 rows=151905
width=12) (actual time=10706.970..10894.299 rows=165481 loops=1)
```

```
                                Sort Key: orders.o_orderkey
                                ->  Hash Join  (cost=6104.51..74694.50
rows=151905 width=12) (actual time=474.323..10013.645 rows=146583 loops=1)
                                      Hash Cond: ("outer".o_custkey =
"inner".c_custkey)
                                      ->  Seq Scan on orders
(cost=0.00..48027.86 rows=735005 width=16) (actual time=0.048..4196.072 rows=724759
loops=1)
                                            Filter: (o_orderdate <
'1995-03-11'::date)
                                      ->  Hash  (cost=5905.01..5905.01
rows=31001 width=4) (actual time=474.039..474.039 rows=0 loops=1)
                                            ->  Seq Scan on customer
(cost=0.00..5905.01 rows=31001 width=4) (actual time=0.038..433.554 rows=30142
loops=1)
                                                  Filter: (c_mktsegment =
'BUILDING'::bpchar)
 Total runtime: 25125.853 ms
(20 rows)
```

sort_mem 設定後の実行計画をリスト 2.3-5 に示す。

リスト 2.3-5　PostgreSQL8.0 での sort_mem 設定後の実行計画

```
Limit  (cost=328971.19..328971.22 rows=10 width=20) (actual
time=17676.920..17676.938 rows=10 loops=1)
   ->  Sort  (cost=328971.19..329796.24 rows=330017 width=20) (actual
time=17676.914..17676.920 rows=10 loops=1)
         Sort Key: sum((lineitem.l_extendedprice * (1::double precision -
lineitem.l_discount))), orders.o_orderdate
         ->  HashAggregate  (cost=296246.41..298721.54 rows=330017 width=20)
(actual time=17617.762..17642.498 rows=11613 loops=1)
               ->  Hash Join  (cost=59584.21..292946.24 rows=330017 width=20)
(actual time=3117.829..17523.454 rows=30511 loops=1)
                     Hash Cond: ("outer".l_orderkey = "inner".o_orderkey)
                     ->  Seq Scan on lineitem  (cost=0.00..205623.83 rows=3258404
width=12) (actual time=0.053..7873.842 rows=3251845 loops=1)
                           Filter: (l_shipdate > '1995-03-11'::date)
                     ->  Hash  (cost=59204.45..59204.45 rows=151905 width=12)
```

```
(actual time=3115.316..3115.316 rows=0 loops=1)
                          ->  Hash Join  (cost=5982.51..59204.45 rows=151905
width=12) (actual time=238.560..2924.907 rows=146583 loops=1)
                                Hash Cond: ("outer".o_custkey =
"inner".c_custkey)
                                ->  Seq Scan on orders  (cost=0.00..48027.86
rows=735005 width=16) (actual time=0.054..1757.734 rows=724759 loops=1)
                                      Filter: (o_orderdate < '1995-03-11'::date)
                                ->  Hash  (cost=5905.01..5905.01 rows=31001
width=4) (actual time=237.063..237.063 rows=0 loops=1)
                                      ->  Seq Scan on customer
(cost=0.00..5905.01 rows=31001 width=4) (actual time=0.109..204.420 rows=30142
loops=1)
                                            Filter: (c_mktsegment =
'BUILDING'::bpchar)
 Total runtime: 17691.942 ms
(17 rows)
```

2.4 Q4

問い合わせをリスト 2.4-1 に示す。

リスト 2.4-1　Q4 の問い合わせ

```
select o_orderpriority, count(*) as order_count from orders where o_orderdate >= date
'1994-09-01' and o_orderdate < date '1994-09-01' + interval '3 month' and exists
( select * from lineitem where l_orderkey = o_orderkey and l_commitdate <
l_receiptdate ) group by o_orderpriority order by o_orderpriority;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.4-2 に示す。

リスト 2.4-2　Q4 の実行計画

```
Sort  (cost=2349727.11..2349727.12 rows=1 width=19) (actual
time=3638.790..3638.792 rows=5 loops=1)
   Sort Key: o_orderpriority
   ->  HashAggregate  (cost=2349727.10..2349727.10 rows=1 width=19) (actual
time=3638.762..3638.768 rows=5 loops=1)
```

```
        ->  Seq Scan on orders  (cost=0.00..2348975.69 rows=150282 width=19)
(actual time=2.889..3495.486 rows=51726 loops=1)
              Filter: ((o_orderdate >= '1994-09-01'::date) AND
((o_orderdate)::timestamp without time zone < '1994-12-01 00:00:00'::timestamp
without time zone) AND (subplan))
              SubPlan
                ->  Index Scan using i_l_orderkey on lineitem  (cost=0.00..3.06
rows=2 width=127) (actual time=0.027..0.027 rows=1 loops=56544)
                      Index Cond: (l_orderkey = $0)
                      Filter: (l_commitdate < l_receiptdate)
 Total runtime: 3639.197 ms
(10 rows)
```

ここでは(o_orderdate >= date '1994-09-01' and o_orderdate < date '1994-09-01' + interval '3 month')の検索にインデックスが使われていないのが気になるが、実際には(o_orderdate >= date '1994-09-01' and o_orderdate < (date '1994-09-01' + interval '3 month')::date)と書き換えてインデックスを使用させるようにしても実行時間の大きな変化はなかった。

PostgreSQL8.0 では、問い合わせを書き換えなくてもインデックスが使われる実行計画になっている。

リスト 2.4-3　PostgreSQL8.0 での実行計画

```
Sort  (cost=214345.08..214345.09 rows=1 width=19) (actual time=3963.295..3963.297
rows=5 loops=1)
   Sort Key: o_orderpriority
   ->  HashAggregate  (cost=214345.07..214345.07 rows=1 width=19) (actual
time=3963.264..3963.272 rows=5 loops=1)
         ->  Index Scan using i_o_orderdate on orders  (cost=0.00..214212.16
rows=26582 width=19) (actual time=12.583..3775.767 rows=51698 loops=1)
               Index Cond: ((o_orderdate >= '1994-09-01'::date) AND (o_orderdate <
'1994-12-01 00:00:00'::timestamp without time zone))
               Filter: (subplan)
               SubPlan
                 ->  Index Scan using i_l_orderkey on lineitem  (cost=0.00..8.85
rows=60 width=127) (actual time=0.044..0.044 rows=1 loops=56516)
                       Index Cond: (l_orderkey = $0)
                       Filter: (l_commitdate < l_receiptdate)
 Total runtime: 3963.650 ms
```

```
(11 rows)
```

2.5 Q5

問い合わせをリスト 2.5-1 に示す。

リスト 2.5-1　Q5 の問い合わせ

```
select n_name, sum(l_extendedprice * (1 - l_discount)) as revenue from customer,
orders, lineitem, supplier, nation, region where c_custkey = o_custkey and l_orderkey
= o_orderkey and l_suppkey = s_suppkey and c_nationkey = s_nationkey and s_nationkey
= n_nationkey and n_regionkey = r_regionkey and r_name = 'AFRICA' and o_orderdate
>= date '1995-01-01' and o_orderdate < date '1995-01-01' + interval '1 year' group
by n_name order by revenue desc;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.5-2 に示す。

リスト 2.5-2　Q5 の実行計画

```
Sort  (cost=329114.93..329114.99 rows=25 width=37) (actual
time=22774.011..22774.014 rows=5 loops=1)
  Sort Key: sum((lineitem.l_extendedprice * (1::double precision -
lineitem.l_discount)))
  ->  HashAggregate  (cost=329114.16..329114.35 rows=25 width=37) (actual
time=22773.978..22773.989 rows=5 loops=1)
        ->  Hash Join  (cost=81360.41..329033.08 rows=16215 width=37) (actual
time=3969.262..22726.043 rows=7203 loops=1)
              Hash Cond: (("outer".c_nationkey = "inner".s_nationkey) AND
("outer".l_suppkey = "inner".s_suppkey))
              ->  Merge Join  (cost=80928.84..308716.29 rows=405369 width=49)
(actual time=3934.204..21918.464 rows=182777 loops=1)
                    Merge Cond: ("outer".l_orderkey = "inner".o_orderkey)
                    ->  Index Scan using i_l_orderkey on lineitem
(cost=0.00..206716.32 rows=5996302 width=16) (actual time=0.115..11749.502
rows=6000828 loops=1)
                    ->  Sort  (cost=80928.84..81205.27 rows=110572 width=41)
(actual time=3933.672..4133.833 rows=182774 loops=1)
                          Sort Key: orders.o_orderkey
```

```
                          ->  Hash Join  (cost=7583.43..69056.26 rows=110572
width=41) (actual time=434.002..3585.151 rows=45839 loops=1)
                                Hash Cond: ("outer".o_custkey =
"inner".c_custkey)
                                ->  Seq Scan on orders  (cost=0.00..54783.28
rows=276426 width=8) (actual time=2.697..1896.401 rows=228673 loops=1)
                                      Filter: ((o_orderdate >=
'1995-01-01'::date) AND ((o_orderdate)::timestamp without time zone < '1996-01-01
00:00:00'::timestamp without time zone))
                                ->  Hash  (cost=6869.33..6869.33 rows=66040
width=41) (actual time=429.814..429.814 rows=0 loops=1)
                                      ->  Hash Join  (cost=2.58..6869.33
rows=66040 width=41) (actual time=0.220..376.875 rows=29764 loops=1)
                                            Hash Cond: ("outer".c_nationkey =
"inner".n_nationkey)
                                            ->  Seq Scan on customer
(cost=0.00..5455.90 rows=150090 width=8) (actual time=0.026..203.444 rows=150000
loops=1)
                                            ->  Hash  (cost=2.55..2.55 rows=11
width=33) (actual time=0.164..0.164 rows=0 loops=1)
                                                  ->  Hash Join
(cost=1.07..2.55 rows=11 width=33) (actual time=0.107..0.158 rows=5 loops=1)
                                                        Hash Cond:
("outer".n_regionkey = "inner".r_regionkey)
                                                        ->  Seq Scan on nation
(cost=0.00..1.25 rows=25 width=37) (actual time=0.024..0.048 rows=25 loops=1)
                                                        ->  Hash
(cost=1.06..1.06 rows=2 width=4) (actual time=0.048..0.048 rows=0 loops=1)
                                                              ->  Seq Scan on
region  (cost=0.00..1.06 rows=2 width=4) (actual time=0.039..0.044 rows=1 loops=1)
                                                                    Filter:
(r_name = 'AFRICA'::bpchar)
                    ->  Hash  (cost=341.38..341.38 rows=10038 width=8) (actual
time=27.513..27.513 rows=0 loops=1)
                          ->  Seq Scan on supplier  (cost=0.00..341.38 rows=10038
width=8) (actual time=0.045..16.157 rows=10000 loops=1)
 Total runtime: 22778.151 ms
(28 rows)
```

特に改善する余地はなかった。

PostgreSQL 8.0 の実行計画をリスト 2.5-3 に示す。7.4 との大きな違いはない。こちらも特に改善の余地はなかった。

リスト 2.5-3　PostgreSQL8.0 での実行計画

```
Sort  (cost=303767.54..303767.60 rows=25 width=37) (actual
time=20625.012..20625.015 rows=5 loops=1)
  Sort Key: sum((lineitem.l_extendedprice * (1::double precision -
lineitem.l_discount)))
  ->  HashAggregate  (cost=303766.77..303766.96 rows=25 width=37) (actual
time=20624.980..20624.990 rows=5 loops=1)
        ->  Hash Join  (cost=67280.10..303730.93 rows=7169 width=37) (actual
time=4105.891..20568.875 rows=7207 loops=1)
              Hash Cond: (("outer".c_nationkey = "inner".s_nationkey) AND
("outer".l_suppkey = "inner".s_suppkey))
              ->  Merge Join  (cost=66840.10..294296.52 rows=179224 width=49)
(actual time=4077.037..19823.082 rows=182687 loops=1)
                    Merge Cond: ("outer".l_orderkey = "inner".o_orderkey)
                    ->  Index Scan using i_l_orderkey on lineitem
(cost=0.00..209763.73 rows=6002066 width=16) (actual time=0.088..9919.403
rows=6001108 loops=1)
                    ->  Sort  (cost=66840.10..66952.07 rows=44786 width=41)
(actual time=4076.813..4273.000 rows=182684 loops=1)
                          Sort Key: orders.o_orderkey
                          ->  Hash Join  (cost=6921.53..62500.47 rows=44786
width=41) (actual time=431.381..3755.080 rows=45822 loops=1)
                                Hash Cond: ("outer".o_custkey =
"inner".c_custkey)
                                ->  Seq Scan on orders  (cost=0.00..51777.43
rows=223927 width=8) (actual time=0.050..1693.433 rows=228637 loops=1)
                                      Filter: ((o_orderdate >=
'1995-01-01'::date) AND (o_orderdate < '1996-01-01 00:00:00'::timestamp without time
zone))
                                ->  Hash  (cost=6582.53..6582.53 rows=30001
width=41) (actual time=430.640..430.640 rows=0 loops=1)
                                      ->  Hash Join  (cost=2.50..6582.53
rows=30001 width=41) (actual time=0.203..379.373 rows=29764 loops=1)
                                            Hash Cond: ("outer".c_nationkey =
"inner".n_nationkey)
```

```
                                          ->  Seq Scan on customer
(cost=0.00..5530.01 rows=150001 width=8) (actual time=0.025..211.921 rows=150000
loops=1)
                                          ->  Hash  (cost=2.49..2.49 rows=5
width=33) (actual time=0.156..0.156 rows=0 loops=1)
                                                ->  Hash Join
(cost=1.06..2.49 rows=5 width=33) (actual time=0.090..0.141 rows=5 loops=1)
                                                      Hash Cond:
("outer".n_regionkey = "inner".r_regionkey)
                                                      ->  Seq Scan on nation
(cost=0.00..1.25 rows=25 width=37) (actual time=0.005..0.031 rows=25 loops=1)
                                                      ->  Hash
(cost=1.06..1.06 rows=1 width=4) (actual time=0.060..0.060 rows=0 loops=1)
                                                            ->  Seq Scan on
region  (cost=0.00..1.06 rows=1 width=4) (actual time=0.043..0.049 rows=1 loops=1)
                                                                  Filter:
(r_name = 'AFRICA'::bpchar)
              ->  Hash  (cost=346.00..346.00 rows=10000 width=8) (actual
time=24.427..24.427 rows=0 loops=1)
                    ->  Seq Scan on supplier  (cost=0.00..346.00 rows=10000
width=8) (actual time=0.014..13.447 rows=10000 loops=1)
 Total runtime: 20629.514 ms
(28 rows)
```

2.6 Q6

問い合わせをリスト 2.6-1 に示す。

リスト 2.6-1　Q6 の問い合わせ

```
select sum(l_extendedprice * l_discount) as revenue from lineitem where l_shipdate
>= date '1995-01-01' and l_shipdate < date '1995-01-01' + interval '1 year' and
l_discount between 0.04 - 0.01 and 0.04 + 0.01 and l_quantity < 25;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.6-2 に示す。

リスト 2.6-2　Q6 の実行計画

```
Aggregate  (cost=277803.14..277803.14 rows=1 width=8) (actual
time=8148.153..8148.153 rows=1 loops=1)
   ->  Seq Scan on lineitem  (cost=0.00..277531.55 rows=108633 width=8) (actual
time=11.979..8083.535 rows=40151 loops=1)
         Filter: ((l_shipdate >= '1995-01-01'::date) AND ((l_shipdate)::timestamp
without time zone < '1996-01-01 00:00:00'::timestamp without time zone) AND
(l_discount >= 0.03::double precision) AND (l_discount <= 0.05::double precision)
AND (l_quantity < 25::double precision))
 Total runtime: 8148.215 ms
(4 rows)
```

これは、lineitem テーブルだけにアクセスする単純な問い合わせである。WHERE 句の条件は l_shipdate と l_discount, それに l_quantity に対するものになっている。それぞれの列にはインデックスが設定してあるが、個々の列に対しては選択されるデータが多すぎるため、インデックスが使用されない。そこで、この 3 つの列の組合わせに対してインデックスを設定してみる。

```
create index i_l_shipdate_discount_quantity on
lineitem(l_shipdate,l_discount,l_quantity);
```

また、l_shipdate は date 型、l_discount と l_quantity は real 型であるのに対し、問い合わせ中の(date '1995-01-01' + interval '1 year')が timestamp without time zone 型、(0.04 - 0.01)と(0.04 + 0.01)、それに 25 は double precision 型と解釈されて型が合わないため、キャストを使用して問い合わせを書き換える(リスト 2.6-3)。

リスト 2.6-3　書き換え後の問い合わせ

```
select sum(l_extendedprice * l_discount) as revenue from lineitem where l_shipdate
>= date '1995-01-01' and l_shipdate < (date '1995-01-01' + interval '1 year')::date
and l_discount between (0.04 - 0.01)::real and (0.04 + 0.01)::real and l_quantity
< 25::real;
```

更に強制的にインデックスを使用させるために set enable_seqscan to off にしたが、結果として改善は見られなかった。

PostgreSQL 8.0 で、インデックスを設定する前の実行計画をリスト 2.6-4 に示す。

リスト 2.6-4　PostgreSQL8.0 におけるインデックス設定前の実行計画

```
Aggregate  (cost=265865.26..265865.27 rows=1 width=8) (actual
time=7287.825..7287.826 rows=1 loops=1)
   ->  Seq Scan on lineitem  (cost=0.00..265644.48 rows=88310 width=8) (actual
time=0.063..7231.228 rows=40135 loops=1)
         Filter: ((l_shipdate >= '1995-01-01'::date) AND (l_shipdate < '1996-01-01
00:00:00'::timestamp without time zone) AND (l_discount >= 0.03::double precision)
AND (l_discount <= 0.05::double precision) AND (l_quantity < 25::double precision))
 Total runtime: 7288.020 ms
(4 rows)
```

7.4 同様にインデックスを設定、更に random_page_cost を 2 に設定してインデックスを使用させるとかなり改善が見られた(リスト 2.6-5)。

```
create index i_l_shipdate_discount_quantity on
lineitem(l_shipdate,l_discount,l_quantity);
set random_page_cost to 2;
```

リスト 2.6-5　PostgreSQL8.0 におけるインデックス設定後の実行計画

```
Aggregate  (cost=177811.42..177811.42 rows=1 width=8) (actual
time=1131.220..1131.220 rows=1 loops=1)
   ->  Index Scan using i_l_shipdate_discount_quantity on lineitem
(cost=0.00..177590.68 rows=88297 width=8) (actual time=0.148..1076.306 rows=40135
loops=1)
         Index Cond: ((l_shipdate >= '1995-01-01'::date) AND (l_shipdate <
'1996-01-01 00:00:00'::timestamp without time zone) AND (l_discount >= 0.03::double
precision) AND (l_discount <= 0.05::double precision) AND (l_quantity < 25::double
precision))
 Total runtime: 1131.302 ms
(4 rows)
```

2.7 Q7

問い合わせをリスト 2.7-1 に示す。

リスト 2.7-1　Q7 の問い合わせ

```
select supp_nation, cust_nation, l_year, sum(volume) as revenue from ( select
```

```
n1.n_name as supp_nation, n2.n_name as cust_nation, extract(year from l_shipdate)
as l_year, l_extendedprice * (1 - l_discount) as volume from supplier, lineitem,
orders, customer, nation n1, nation n2 where s_suppkey = l_suppkey and o_orderkey
= l_orderkey and c_custkey = o_custkey and s_nationkey = n1.n_nationkey and
c_nationkey = n2.n_nationkey and ( (n1.n_name = 'INDIA' and n2.n_name = 'CHINA') or
(n1.n_name = 'CHINA' and n2.n_name = 'INDIA') ) and l_shipdate between date
'1995-01-01' and date '1996-12-31' ) as shipping group by supp_nation, cust_nation,
l_year order by supp_nation, cust_nation, l_year;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.7-2 に示す。

リスト 2.7-2　Q7 の実行計画

```
GroupAggregate  (cost=74848.82..74850.47 rows=66 width=70) (actual
time=25648.148..25663.595 rows=4 loops=1)
   ->  Sort  (cost=74848.82..74848.99 rows=66 width=70) (actual
time=25643.056..25649.469 rows=6111 loops=1)
         Sort Key: n1.n_name, n2.n_name, date_part('year'::text,
(lineitem.l_shipdate)::timestamp without time zone)
         ->  Merge Join  (cost=74409.64..74846.83 rows=66 width=70) (actual
time=25174.932..25572.066 rows=6111 loops=1)
               Merge Cond: ("outer".s_suppkey = "inner".l_suppkey)
               Join Filter: ("outer".s_nationkey = "inner".n_nationkey)
               ->  Index Scan using supplier_pkey on supplier  (cost=0.00..383.38
rows=10038 width=8) (actual time=0.120..22.322 rows=10000 loops=1)
               ->  Sort  (cost=74409.64..74413.74 rows=1642 width=78) (actual
time=25174.354..25325.125 rows=146062 loops=1)
                     Sort Key: lineitem.l_suppkey
                     ->  Nested Loop  (cost=6352.27..74321.94 rows=1642 width=78)
(actual time=370.708..20331.046 rows=146062 loops=1)
                           ->  Hash Join  (cost=6352.27..69773.32 rows=1476
width=66) (actual time=367.380..15384.290 rows=120628 loops=1)
                                 Hash Cond: ("outer".o_custkey =
"inner".c_custkey)
                                 ->  Seq Scan on orders  (cost=0.00..43532.30
rows=1500130 width=8) (actual time=0.196..1911.745 rows=1500000 loops=1)
                                 ->  Hash  (cost=6269.26..6269.26 rows=6004
width=66) (actual time=366.627..366.627 rows=0 loops=1)
```

```
                                    ->  Hash Join  (cost=2.87..6269.26
rows=6004 width=66) (actual time=0.171..343.638 rows=12066 loops=1)
                                          Hash Cond: ("outer".c_nationkey =
"inner".n_nationkey)
                                          ->  Seq Scan on customer
(cost=0.00..5455.90 rows=150090 width=8) (actual time=0.023..197.619 rows=150000
loops=1)
                                          ->  Hash  (cost=2.87..2.87 rows=1
width=66) (actual time=0.123..0.123 rows=0 loops=1)
                                                ->  Nested Loop
(cost=1.38..2.87 rows=1 width=66) (actual time=0.084..0.118 rows=2 loops=1)
                                                      Join Filter:
((("inner".n_name = 'INDIA'::bpchar) OR ("outer".n_name = 'INDIA'::bpchar)) AND
(("outer".n_name = 'CHINA'::bpchar) OR ("inner".n_name = 'CHINA'::bpchar)))
                                                      ->  Seq Scan on nation n1
(cost=0.00..1.38 rows=2 width=33) (actual time=0.042..0.058 rows=2 loops=1)
                                                            Filter: ((n_name =
'CHINA'::bpchar) OR (n_name = 'INDIA'::bpchar))
                                                      ->  Materialize
(cost=1.38..1.40 rows=2 width=33) (actual time=0.010..0.021 rows=2 loops=2)
                                                            ->  Seq Scan on
nation n2  (cost=0.00..1.38 rows=2 width=33) (actual time=0.014..0.030 rows=2
loops=1)
                                                                  Filter:
((n_name = 'INDIA'::bpchar) OR (n_name = 'CHINA'::bpchar))
                              ->  Index Scan using i_l_orderkey on lineitem
(cost=0.00..3.07 rows=1 width=20) (actual time=0.031..0.034 rows=1 loops=120628)
                                    Index Cond: ("outer".o_orderkey =
lineitem.l_orderkey)
                                    Filter: ((l_shipdate >= '1995-01-01'::date) AND
(l_shipdate <= '1996-12-31'::date))
 Total runtime: 25686.823 ms
(29 rows)
```

この問い合わせでは、改善の余地は見いだせなかった。

PostgreSQL 8.0 での問い合わせ計画をリスト 2.7-3 に示す。7.4 同様改善の余地は見いだせなかった。

リスト 2.7-3　PostgreSQL8.0 での問い合わせ計画

```
GroupAggregate  (cost=333319.35..333466.37 rows=5881 width=70) (actual
time=21971.929..21989.525 rows=4 loops=1)
   ->  Sort  (cost=333319.35..333334.05 rows=5881 width=70) (actual
time=21966.135..21972.586 rows=6100 loops=1)
         Sort Key: n1.n_name, n2.n_name, date_part('year'::text,
(lineitem.l_shipdate)::timestamp without time zone)
         ->  Hash Join  (cost=267659.78..332951.14 rows=5881 width=70) (actual
time=13877.264..21851.627 rows=6100 loops=1)
               Hash Cond: (("outer".o_custkey = "inner".c_custkey) AND
("outer".n_nationkey = "inner".c_nationkey))
               ->  Merge Join  (cost=260719.76..316974.25 rows=147022 width=78)
(actual time=13359.043..19987.831 rows=150106 loops=1)
                     Merge Cond: ("outer".o_orderkey = "inner".l_orderkey)
                     ->  Index Scan using orders_pkey on orders
(cost=0.00..50353.12 rows=1499829 width=8) (actual time=14.524..4490.591
rows=1499996 loops=1)
                     ->  Sort  (cost=260719.76..261087.32 rows=147022 width=78)
(actual time=13344.484..13525.520 rows=150106 loops=1)
                           Sort Key: lineitem.l_orderkey
                           ->  Hash Join  (cost=427.25..231711.11 rows=147022
width=78) (actual time=40.097..12011.545 rows=150106 loops=1)
                                 Hash Cond: ("outer".l_suppkey =
"inner".s_suppkey)
                                 ->  Seq Scan on lineitem  (cost=0.00..220616.23
rows=1839245 width=20) (actual time=0.058..9042.130 rows=1828450 loops=1)
                                       Filter: ((l_shipdate >= '1995-01-01'::date)
AND (l_shipdate <= '1996-12-31'::date))
                                 ->  Hash  (cost=425.25..425.25 rows=800
width=66) (actual time=39.991..39.991 rows=0 loops=1)
                                       ->  Hash Join  (cost=21.25..425.25
rows=800 width=66) (actual time=1.399..38.898 rows=822 loops=1)
                                             Hash Cond: ("outer".s_nationkey =
"inner".n_nationkey)
                                             ->  Seq Scan on supplier
(cost=0.00..346.00 rows=10000 width=8) (actual time=0.029..27.666 rows=10000
loops=1)
                                             ->  Hash  (cost=21.25..21.25 rows=2
width=66) (actual time=1.331..1.331 rows=0 loops=1)
```

```
                                                    ->  Nested Loop
(cost=1.25..21.25 rows=2 width=66) (actual time=0.523..1.316 rows=2 loops=1)
                                                          Join Filter:
((("outer".n_name = 'INDIA'::bpchar) AND ("inner".n_name = 'CHINA'::bpchar)) OR
(("outer".n_name = 'CHINA'::bpchar) AND ("inner".n_name = 'INDIA'::bpchar)))
                                                          ->  Seq Scan on nation n1
(cost=0.00..1.25 rows=25 width=33) (actual time=0.007..0.032 rows=25 loops=1)
                                                          ->  Materialize
(cost=1.25..1.50 rows=25 width=33) (actual time=0.001..0.019 rows=25 loops=25)
                                                                ->  Seq Scan on
nation n2  (cost=0.00..1.25 rows=25 width=33) (actual time=0.006..0.040 rows=25
loops=1)
               ->  Hash  (cost=5530.01..5530.01 rows=150001 width=8) (actual
time=512.489..512.489 rows=0 loops=1)
                     ->  Seq Scan on customer  (cost=0.00..5530.01 rows=150001
width=8) (actual time=0.045..309.879 rows=150000 loops=1)
 Total runtime: 22007.420 ms
(27 rows)
```

2.8 Q8

問い合わせをリスト 2.8-1 に示す。

リスト 2.8-1　Q8 の問い合わせ

```
select o_year, sum(case when nation = 'CHINA' then volume else 0 end) / sum(volume)
as mkt_share from ( select extract(year from o_orderdate) as o_year, l_extendedprice
* (1 - l_discount) as volume, n2.n_name as nation from part, supplier, lineitem,
orders, customer, nation n1, nation n2, region where p_partkey = l_partkey and
s_suppkey = l_suppkey and l_orderkey = o_orderkey and o_custkey = c_custkey and
c_nationkey = n1.n_nationkey and n1.n_regionkey = r_regionkey and r_name = 'ASIA'
and s_nationkey = n2.n_nationkey and o_orderdate between date '1995-01-01' and date
'1996-12-31' and p_type = 'PROMO PLATED STEEL' ) as all_nations group by o_year order
by o_year;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.8-2 に示す。

リスト 2.8-2　Q8 の実行計画

```
Sort  (cost=241342.75..241344.64 rows=756 width=41) (actual
time=7273.505..7273.506 rows=2 loops=1)
   Sort Key: date_part('year'::text, (orders.o_orderdate)::timestamp without time
zone)
   ->  HashAggregate  (cost=241287.71..241306.61 rows=756 width=41) (actual
time=7273.441..7273.449 rows=2 loops=1)
         ->  Hash Join  (cost=240594.74..241254.73 rows=4397 width=41) (actual
time=7220.935..7265.785 rows=2356 loops=1)
               Hash Cond: ("outer".s_nationkey = "inner".n_nationkey)
               ->  Hash Join  (cost=240593.43..241154.46 rows=4398 width=16)
(actual time=7220.794..7255.666 rows=2356 loops=1)
                     Hash Cond: ("outer".s_suppkey = "inner".l_suppkey)
                     ->  Seq Scan on supplier  (cost=0.00..341.38 rows=10038
width=8) (actual time=0.047..13.171 rows=10000 loops=1)
                     ->  Hash  (cost=240582.44..240582.44 rows=4397 width=16)
(actual time=7220.695..7220.695 rows=0 loops=1)
                           ->  Hash Join  (cost=233875.14..240582.44 rows=4397
width=16) (actual time=6696.892..7216.436 rows=2356 loops=1)
                                 Hash Cond: ("outer".c_nationkey =
"inner".n_nationkey)
                                 ->  Merge Join  (cost=233872.56..240476.53
rows=10991 width=20) (actual time=6696.439..7200.915 rows=11621 loops=1)
                                       Merge Cond: ("outer".c_custkey =
"inner".o_custkey)
                                       ->  Index Scan using customer_pkey on
customer  (cost=0.00..6063.90 rows=150090 width=8) (actual time=0.047..307.186
rows=149999 loops=1)
                                       ->  Sort  (cost=233872.56..233900.04
rows=10990 width=20) (actual time=6696.326..6709.495 rows=11621 loops=1)
                                             Sort Key: orders.o_custkey
                                             ->  Merge Join
(cost=174563.46..233134.92 rows=10990 width=20) (actual time=2959.511..6626.961
rows=11621 loops=1)
                                                   Merge Cond:
("outer".o_orderkey = "inner".l_orderkey)
                                                   ->  Index Scan using
orders_pkey on orders  (cost=0.00..57107.95 rows=466708 width=12) (actual
time=11.621..3049.470 rows=457295 loops=1)
```

```
                                                    Filter: ((o_orderdate >=
'1995-01-01'::date) AND (o_orderdate <= '1996-12-31'::date))
                                                ->  Sort
(cost=174563.46..174659.75 rows=38515 width=16) (actual time=2947.362..2993.004
rows=38137 loops=1)
                                                    Sort Key:
lineitem.l_orderkey
                                                    ->  Nested Loop
(cost=0.00..171245.19 rows=38515 width=16) (actual time=3.384..2668.855 rows=38137
loops=1)
                                                        ->  Seq Scan on
part  (cost=0.00..7257.77 rows=1285 width=4) (actual time=0.045..220.215 rows=1266
loops=1)
                                                            Filter:
((p_type)::text = 'PROMO PLATED STEEL'::text)
                                                        ->  Index Scan
using i_l_partkey on lineitem  (cost=0.00..127.23 rows=31 width=20) (actual
time=1.283..1.856 rows=30 loops=1266)
                                                            Index Cond:
("outer".p_partkey = lineitem.l_partkey)
                                    ->  Hash  (cost=2.55..2.55 rows=11 width=4)
(actual time=0.116..0.116 rows=0 loops=1)
                                        ->  Hash Join  (cost=1.07..2.55 rows=11
width=4) (actual time=0.075..0.111 rows=5 loops=1)
                                            Hash Cond: ("outer".n_regionkey =
"inner".r_regionkey)
                                            ->  Seq Scan on nation n1
(cost=0.00..1.25 rows=25 width=8) (actual time=0.002..0.025 rows=25 loops=1)
                                            ->  Hash  (cost=1.06..1.06 rows=2
width=4) (actual time=0.044..0.044 rows=0 loops=1)
                                                ->  Seq Scan on region
(cost=0.00..1.06 rows=2 width=4) (actual time=0.037..0.040 rows=1 loops=1)
                                                    Filter: (r_name =
'ASIA'::bpchar)
                  ->  Hash  (cost=1.25..1.25 rows=25 width=33) (actual
time=0.091..0.091 rows=0 loops=1)
                      ->  Seq Scan on nation n2  (cost=0.00..1.25 rows=25 width=33)
(actual time=0.032..0.064 rows=25 loops=1)
 Total runtime: 7277.037 ms
(37 rows)
```

sort_mem を増やし(set sort_mem to 102400)、merge join を hash join にすることで高速化できた(リスト 2.8-3)。

リスト 2.8-3　sort_mem を増やした後の実行計画

```
Sort  (cost=226169.26..226171.15 rows=754 width=41) (actual
time=4086.140..4086.141 rows=2 loops=1)
   Sort Key: date_part('year'::text, (orders.o_orderdate)::timestamp without time
zone)
   ->  HashAggregate  (cost=226114.38..226133.23 rows=754 width=41) (actual
time=4086.119..4086.126 rows=2 loops=1)
         ->  Hash Join  (cost=225425.38..226081.95 rows=4324 width=41) (actual
time=4042.561..4078.853 rows=2351 loops=1)
               Hash Cond: ("outer".s_nationkey = "inner".n_nationkey)
               ->  Hash Join  (cost=225424.07..225983.32 rows=4325 width=16)
(actual time=4042.086..4069.088 rows=2351 loops=1)
                     Hash Cond: ("outer".s_suppkey = "inner".l_suppkey)
                     ->  Seq Scan on supplier  (cost=0.00..341.00 rows=10000
width=8) (actual time=0.042..13.419 rows=10000 loops=1)
                     ->  Hash  (cost=225413.26..225413.26 rows=4324 width=16)
(actual time=4041.032..4041.032 rows=0 loops=1)
                           ->  Hash Join  (cost=158410.12..225413.26 rows=4324
width=16) (actual time=1514.896..4035.556 rows=2351 loops=1)
                                 Hash Cond: ("outer".o_orderkey =
"inner".l_orderkey)
                                 ->  Hash Join  (cost=7032.59..62447.86
rows=184714 width=8) (actual time=401.393..2810.806 rows=91783 loops=1)
                                       Hash Cond: ("outer".o_custkey =
"inner".c_custkey)
                                       ->  Seq Scan on orders
(cost=0.00..51074.50 rows=461780 width=12) (actual time=0.054..1817.472 rows=457707
loops=1)
                                             Filter: ((o_orderdate >=
'1995-01-01'::date) AND (o_orderdate <= '1996-12-31'::date))
                                       ->  Hash  (cost=6867.59..6867.59
rows=66001 width=4) (actual time=400.014..400.014 rows=0 loops=1)
                                             ->  Hash Join  (cost=2.58..6867.59
rows=66001 width=4) (actual time=2.930..368.178 rows=30183 loops=1)
```

```
                                                    Hash Cond:
("outer".c_nationkey = "inner".n_nationkey)
                                                    ->  Seq Scan on customer
(cost=0.00..5455.00 rows=150000 width=8) (actual time=0.027..203.348 rows=150000
loops=1)
                                                    ->  Hash  (cost=2.55..2.55
rows=11 width=4) (actual time=1.561..1.561 rows=0 loops=1)
                                                          ->  Hash Join
(cost=1.07..2.55 rows=11 width=4) (actual time=1.507..1.544 rows=5 loops=1)
                                                                Hash Cond:
("outer".n_regionkey = "inner".r_regionkey)
                                                                ->  Seq Scan on
nation n1  (cost=0.00..1.25 rows=25 width=8) (actual time=0.029..0.052 rows=25
loops=1)
                                                                ->  Hash
(cost=1.06..1.06 rows=2 width=4) (actual time=0.106..0.106 rows=0 loops=1)
                                                                      ->  Seq Scan
on region  (cost=0.00..1.06 rows=2 width=4) (actual time=0.078..0.082 rows=1
loops=1)

Filter: (r_name = 'ASIA'::bpchar)
                                  ->  Hash  (cost=151289.67..151289.67 rows=35143
width=16) (actual time=1111.383..1111.383 rows=0 loops=1)
                                        ->  Nested Loop  (cost=0.00..151289.67
rows=35143 width=16) (actual time=0.207..1044.619 rows=38137 loops=1)
                                              ->  Seq Scan on part
(cost=0.00..7257.00 rows=1292 width=4) (actual time=0.063..211.866 rows=1266
loops=1)
                                                    Filter: ((p_type)::text =
'PROMO PLATED STEEL'::text)
                                              ->  Index Scan using i_l_partkey on
lineitem  (cost=0.00..111.14 rows=27 width=20) (actual time=0.048..0.591 rows=30
loops=1266)
                                                    Index Cond: ("outer".p_partkey
= lineitem.l_partkey)
                    ->  Hash  (cost=1.25..1.25 rows=25 width=33) (actual
time=0.149..0.149 rows=0 loops=1)
                          ->  Seq Scan on nation n2  (cost=0.00..1.25 rows=25 width=33)
(actual time=0.071..0.102 rows=25 loops=1)
 Total runtime: 4089.337 ms
```

```
(35 rows)
```

問い合わせ計画をよく見ると、(p_type = 'PROMO PLATED STEEL')に対応する問い合わせ計画が

```
  ->  Seq Scan on part  (cost=0.00..7257.77 rows=1285 width=4) (actual
time=0.045..220.215 rows=1266 loops=1)
        Filter: ((p_type)::text = 'PROMO PLATED STEEL'::text)
```

となっていて順スキャンが採用されている。これは、part テーブルの p_type 列にインデックスが設定されていないためである。そこで p_type にインデックスを設定した後の問い合わせ計画をリスト 2.8-4 に示す。

```
create index i_p_type on part(p_type);
```

リスト 2.8-4　p_type にインデックス設定後の実行計画

```
Sort  (cost=223487.31..223489.19 rows=754 width=41) (actual
time=3955.447..3955.448 rows=2 loops=1)
   Sort Key: date_part('year'::text, (orders.o_orderdate)::timestamp without time
zone)
   ->  HashAggregate  (cost=223432.42..223451.27 rows=754 width=41) (actual
time=3955.426..3955.433 rows=2 loops=1)
         ->  Hash Join  (cost=222743.42..223399.99 rows=4324 width=41) (actual
time=3908.259..3948.046 rows=2351 loops=1)
               Hash Cond: ("outer".s_nationkey = "inner".n_nationkey)
               ->  Hash Join  (cost=222742.11..223301.36 rows=4325 width=16)
(actual time=3907.786..3934.900 rows=2351 loops=1)
                     Hash Cond: ("outer".s_suppkey = "inner".l_suppkey)
                     ->  Seq Scan on supplier  (cost=0.00..341.00 rows=10000
width=8) (actual time=0.042..13.472 rows=10000 loops=1)
                     ->  Hash  (cost=222731.30..222731.30 rows=4324 width=16)
(actual time=3906.739..3906.739 rows=0 loops=1)
                           ->  Hash Join  (cost=155728.17..222731.30 rows=4324
width=16) (actual time=1347.255..3901.128 rows=2351 loops=1)
                                 Hash Cond: ("outer".o_orderkey =
"inner".l_orderkey)
                                 ->  Hash Join  (cost=7032.59..62447.86
rows=184714 width=8) (actual time=401.082..2841.662 rows=91783 loops=1)
```

```
                                        Hash Cond: ("outer".o_custkey =
"inner".c_custkey)
                                     ->  Seq Scan on orders
(cost=0.00..51074.50 rows=461780 width=12) (actual time=0.055..1831.793 rows=457707
loops=1)
                                           Filter: ((o_orderdate >=
'1995-01-01'::date) AND (o_orderdate <= '1996-12-31'::date))
                                     ->  Hash  (cost=6867.59..6867.59
rows=66001 width=4) (actual time=399.661..399.661 rows=0 loops=1)
                                           ->  Hash Join  (cost=2.58..6867.59
rows=66001 width=4) (actual time=3.041..367.008 rows=30183 loops=1)
                                                 Hash Cond:
("outer".c_nationkey = "inner".n_nationkey)
                                                 ->  Seq Scan on customer
(cost=0.00..5455.00 rows=150000 width=8) (actual time=0.026..203.836 rows=150000
loops=1)
                                                 ->  Hash  (cost=2.55..2.55
rows=11 width=4) (actual time=1.609..1.609 rows=0 loops=1)
                                                       ->  Hash Join
(cost=1.07..2.55 rows=11 width=4) (actual time=1.554..1.591 rows=5 loops=1)
                                                             Hash Cond:
("outer".n_regionkey = "inner".r_regionkey)
                                                             ->  Seq Scan on
nation n1  (cost=0.00..1.25 rows=25 width=8) (actual time=0.027..0.051 rows=25
loops=1)
                                                             ->  Hash
(cost=1.06..1.06 rows=2 width=4) (actual time=0.119..0.119 rows=0 loops=1)
                                                                   ->  Seq Scan
on region  (cost=0.00..1.06 rows=2 width=4) (actual time=0.093..0.096 rows=1
loops=1)

Filter: (r_name = 'ASIA'::bpchar)
                                  ->  Hash  (cost=148607.72..148607.72 rows=35143
width=16) (actual time=944.013..944.013 rows=0 loops=1)
                                     ->  Nested Loop  (cost=0.00..148607.72
rows=35143 width=16) (actual time=0.290..876.618 rows=38137 loops=1)
                                           ->  Index Scan using i_p_type on part
(cost=0.00..4575.04 rows=1292 width=4) (actual time=0.157..31.557 rows=1266
loops=1)
                                                 Index Cond: ((p_type)::text =
```

```
'PROMO PLATED STEEL'::text)
                                          ->  Index Scan using i_l_partkey on
lineitem  (cost=0.00..111.14 rows=27 width=20) (actual time=0.051..0.599 rows=30
loops=1266)
                                                Index Cond: ("outer".p_partkey
= lineitem.l_partkey)
              ->  Hash  (cost=1.25..1.25 rows=25 width=33) (actual
time=0.161..0.161 rows=0 loops=1)
                    ->  Seq Scan on nation n2  (cost=0.00..1.25 rows=25 width=33)
(actual time=0.069..0.100 rows=25 loops=1)
 Total runtime: 3958.685 ms
(35 rows)
```

PostgreSQL 8.0 での問い合わせ実行計画をリスト 2.8-5 に示す。

リスト 2.8-5　PostgreSQL8.0 での実行計画

```
Sort  (cost=247215.72..247217.45 rows=693 width=41) (actual
time=6314.551..6314.553 rows=2 loops=1)
   Sort Key: date_part('year'::text, (orders.o_orderdate)::timestamp without time
zone)
   ->  HashAggregate  (cost=247165.69..247183.02 rows=693 width=41) (actual
time=6314.526..6314.533 rows=2 loops=1)
         ->  Hash Join  (cost=239879.15..247148.65 rows=2272 width=41) (actual
time=5612.819..6305.429 rows=2348 loops=1)
               Hash Cond: ("outer".s_nationkey = "inner".n_nationkey)
               ->  Hash Join  (cost=239877.83..247101.90 rows=2272 width=16)
(actual time=5612.699..6294.492 rows=2348 loops=1)
                     Hash Cond: ("outer".c_nationkey = "inner".n_nationkey)
                     ->  Hash Join  (cost=239875.33..247019.88 rows=11360
width=20) (actual time=5612.228..6280.853 rows=11614 loops=1)
                           Hash Cond: ("outer".l_suppkey = "inner".s_suppkey)
                           ->  Merge Join  (cost=239460.33..246142.88 rows=11360
width=20) (actual time=5586.400..6198.988 rows=11614 loops=1)
                                 Merge Cond: ("outer".c_custkey =
"inner".o_custkey)
                                 ->  Index Scan using customer_pkey on customer
(cost=0.00..6138.01 rows=150001 width=8) (actual time=6.844..440.029 rows=149999
loops=1)
```

```
                                  ->  Sort  (cost=239460.33..239488.73 rows=11360
width=20) (actual time=5579.494..5592.895 rows=11614 loops=1)
                                        Sort Key: orders.o_custkey
                                        ->  Merge Join  (cost=179493.63..238695.14
rows=11360 width=20) (actual time=2648.096..5505.462 rows=11614 loops=1)
                                              Merge Cond: ("outer".o_orderkey =
"inner".l_orderkey)
                                              ->  Index Scan using orders_pkey on
orders  (cost=0.00..57852.26 rows=440148 width=12) (actual time=0.107..2306.415
rows=457248 loops=1)
                                                    Filter: ((o_orderdate >=
'1995-01-01'::date) AND (o_orderdate <= '1996-12-31'::date))
                                              ->  Sort
(cost=179493.63..179590.40 rows=38708 width=16) (actual time=2647.511..2694.912
rows=38106 loops=1)
                                                    Sort Key: lineitem.l_orderkey
                                                    ->  Nested Loop
(cost=0.00..176114.52 rows=38708 width=16) (actual time=13.032..2361.220 rows=38106
loops=1)
                                                          ->  Seq Scan on part
(cost=0.00..7363.00 rows=1290 width=4) (actual time=0.041..272.194 rows=1266
loops=1)
                                                                Filter:
((p_type)::text = 'PROMO PLATED STEEL'::text)
                                                          ->  Index Scan using
i_l_partkey on lineitem  (cost=0.00..130.42 rows=32 width=20) (actual
time=0.986..1.571 rows=30 loops=1266)
                                                                Index Cond:
("outer".p_partkey = lineitem.l_partkey)
                              ->  Hash  (cost=346.00..346.00 rows=10000 width=8)
(actual time=25.723..25.723 rows=0 loops=1)
                                    ->  Seq Scan on supplier  (cost=0.00..346.00
rows=10000 width=8) (actual time=0.027..15.052 rows=10000 loops=1)
                        ->  Hash  (cost=2.49..2.49 rows=5 width=4) (actual
time=0.124..0.124 rows=0 loops=1)
                              ->  Hash Join  (cost=1.06..2.49 rows=5 width=4) (actual
time=0.081..0.117 rows=5 loops=1)
                                    Hash Cond: ("outer".n_regionkey =
"inner".r_regionkey)
                                    ->  Seq Scan on nation n1  (cost=0.00..1.25
```

```
rows=25 width=8) (actual time=0.002..0.027 rows=25 loops=1)
                                  ->  Hash  (cost=1.06..1.06 rows=1 width=4)
(actual time=0.050..0.050 rows=0 loops=1)
                                        ->  Seq Scan on region  (cost=0.00..1.06
rows=1 width=4) (actual time=0.044..0.046 rows=1 loops=1)
                                              Filter: (r_name = 'ASIA'::bpchar)
              ->  Hash  (cost=1.25..1.25 rows=25 width=33) (actual
time=0.072..0.072 rows=0 loops=1)
                    ->  Seq Scan on nation n2  (cost=0.00..1.25 rows=25 width=33)
(actual time=0.009..0.045 rows=25 loops=1)
 Total runtime: 6318.074 ms
(37 rows)
```

7.4 と同様、(set sort_mem to 102400;)でソートメモリを増やしたところ、改善が見られた(リスト 2.8-6)。

リスト 2.8-6　ソートメモリを増やした後の PostgreSQL8.0 での問い合わせ実行計画

```
Sort  (cost=247618.47..247620.20 rows=693 width=41) (actual
time=4167.802..4167.804 rows=2 loops=1)
   Sort Key: date_part('year'::text, (orders.o_orderdate)::timestamp without time
zone)
   ->  HashAggregate  (cost=247568.44..247585.77 rows=693 width=41) (actual
time=4167.781..4167.788 rows=2 loops=1)
         ->  Hash Join  (cost=183821.75..247551.40 rows=2272 width=41) (actual
time=1696.163..4151.630 rows=2348 loops=1)
               Hash Cond: ("outer".s_nationkey = "inner".n_nationkey)
               ->  Hash Join  (cost=183820.44..247504.65 rows=2272 width=16)
(actual time=1695.366..4133.477 rows=2348 loops=1)
                     Hash Cond: ("outer".o_orderkey = "inner".l_orderkey)
                     ->  Hash Join  (cost=6657.53..61516.03 rows=88030 width=8)
(actual time=411.366..2741.213 rows=91707 loops=1)
                           Hash Cond: ("outer".o_custkey = "inner".c_custkey)
                           ->  Seq Scan on orders  (cost=0.00..51777.43
rows=440148 width=12) (actual time=0.058..1746.642 rows=457263 loops=1)
                                 Filter: ((o_orderdate >= '1995-01-01'::date) AND
(o_orderdate <= '1996-12-31'::date))
                           ->  Hash  (cost=6582.53..6582.53 rows=30001 width=4)
(actual time=409.942..409.942 rows=0 loops=1)
```

```
                                ->  Hash Join  (cost=2.50..6582.53 rows=30001
width=4) (actual time=2.938..378.599 rows=30183 loops=1)
                                      Hash Cond: ("outer".c_nationkey =
"inner".n_nationkey)
                                      ->  Seq Scan on customer
(cost=0.00..5530.01 rows=150001 width=8) (actual time=0.031..214.710 rows=150000
loops=1)
                                      ->  Hash  (cost=2.49..2.49 rows=5 width=4)
(actual time=1.520..1.520 rows=0 loops=1)
                                            ->  Hash Join  (cost=1.06..2.49
rows=5 width=4) (actual time=1.466..1.503 rows=5 loops=1)
                                                  Hash Cond:
("outer".n_regionkey = "inner".r_regionkey)
                                                  ->  Seq Scan on nation n1
(cost=0.00..1.25 rows=25 width=8) (actual time=0.006..0.030 rows=25 loops=1)
                                                  ->  Hash  (cost=1.06..1.06
rows=1 width=4) (actual time=0.119..0.119 rows=0 loops=1)
                                                        ->  Seq Scan on region
(cost=0.00..1.06 rows=1 width=4) (actual time=0.092..0.095 rows=1 loops=1)
                                                              Filter: (r_name =
'ASIA'::bpchar)
                          ->  Hash  (cost=177066.14..177066.14 rows=38708 width=16)
(actual time=1281.928..1281.928 rows=0 loops=1)
                                ->  Hash Join  (cost=371.00..177066.14 rows=38708
width=16) (actual time=26.072..1213.761 rows=38106 loops=1)
                                      Hash Cond: ("outer".l_suppkey =
"inner".s_suppkey)
                                      ->  Nested Loop  (cost=0.00..176114.52
rows=38708 width=16) (actual time=0.197..1089.717 rows=38106 loops=1)
                                            ->  Seq Scan on part  (cost=0.00..7363.00
rows=1290 width=4) (actual time=0.070..221.155 rows=1266 loops=1)
                                                  Filter: ((p_type)::text = 'PROMO
PLATED STEEL'::text)
                                            ->  Index Scan using i_l_partkey on lineitem
(cost=0.00..130.42 rows=32 width=20) (actual time=0.053..0.617 rows=30 loops=1266)
                                                  Index Cond: ("outer".p_partkey =
lineitem.l_partkey)
                                      ->  Hash  (cost=346.00..346.00 rows=10000
width=8) (actual time=24.655..24.655 rows=0 loops=1)
                                            ->  Seq Scan on supplier
```

```
(cost=0.00..346.00 rows=10000 width=8) (actual time=0.083..15.006 rows=10000
loops=1)
              ->  Hash  (cost=1.25..1.25 rows=25 width=33) (actual
time=0.095..0.095 rows=0 loops=1)
                    ->  Seq Scan on nation n2  (cost=0.00..1.25 rows=25 width=33)
(actual time=0.019..0.052 rows=25 loops=1)
 Total runtime: 4171.251 ms
(35 rows)
```

8.0 でも part テーブルの p_type テーブルにインデックスを設定することによって若干改善が見られた(リスト 2.8-7)。

```
create index i_p_type on part(p_type);
```

リスト 2.8-7　8.0 で p_type にインデックス設定後の実行計画

```
Sort  (cost=244826.63..244828.36 rows=693 width=41) (actual
time=3981.159..3981.161 rows=2 loops=1)
   Sort Key: date_part('year'::text, (orders.o_orderdate)::timestamp without time
zone)
   ->  HashAggregate  (cost=244776.60..244793.93 rows=693 width=41) (actual
time=3981.137..3981.145 rows=2 loops=1)
         ->  Hash Join  (cost=181029.91..244759.56 rows=2272 width=41) (actual
time=1508.649..3965.122 rows=2348 loops=1)
               Hash Cond: ("outer".s_nationkey = "inner".n_nationkey)
               ->  Hash Join  (cost=181028.60..244712.81 rows=2272 width=16)
(actual time=1507.859..3946.741 rows=2348 loops=1)
                     Hash Cond: ("outer".o_orderkey = "inner".l_orderkey)
                     ->  Hash Join  (cost=6657.53..61516.03 rows=88030 width=8)
(actual time=412.288..2742.520 rows=91707 loops=1)
                           Hash Cond: ("outer".o_custkey = "inner".c_custkey)
                           ->  Seq Scan on orders  (cost=0.00..51777.43
rows=440148 width=12) (actual time=0.059..1749.142 rows=457263 loops=1)
                                 Filter: ((o_orderdate >= '1995-01-01'::date) AND
(o_orderdate <= '1996-12-31'::date))
                           ->  Hash  (cost=6582.53..6582.53 rows=30001 width=4)
(actual time=410.866..410.866 rows=0 loops=1)
                                 ->  Hash Join  (cost=2.50..6582.53 rows=30001
width=4) (actual time=2.927..379.902 rows=30183 loops=1)
```

```
                                    Hash Cond: ("outer".c_nationkey =
"inner".n_nationkey)
                                    ->  Seq Scan on customer
(cost=0.00..5530.01 rows=150001 width=8) (actual time=0.030..214.200 rows=150000
loops=1)
                                    ->  Hash  (cost=2.49..2.49 rows=5 width=4)
(actual time=1.507..1.507 rows=0 loops=1)
                                          ->  Hash Join  (cost=1.06..2.49
rows=5 width=4) (actual time=1.451..1.489 rows=5 loops=1)
                                                Hash Cond:
("outer".n_regionkey = "inner".r_regionkey)
                                                ->  Seq Scan on nation n1
(cost=0.00..1.25 rows=25 width=8) (actual time=0.006..0.033 rows=25 loops=1)
                                                ->  Hash  (cost=1.06..1.06
rows=1 width=4) (actual time=0.122..0.122 rows=0 loops=1)
                                                      ->  Seq Scan on region
(cost=0.00..1.06 rows=1 width=4) (actual time=0.094..0.098 rows=1 loops=1)
                                                            Filter: (r_name =
'ASIA'::bpchar)
                  ->  Hash  (cost=174274.30..174274.30 rows=38708 width=16)
(actual time=1093.517..1093.517 rows=0 loops=1)
                        ->  Hash Join  (cost=371.00..174274.30 rows=38708
width=16) (actual time=26.336..1028.295 rows=38106 loops=1)
                              Hash Cond: ("outer".l_suppkey =
"inner".s_suppkey)
                              ->  Nested Loop  (cost=0.00..173322.68
rows=38708 width=16) (actual time=0.333..902.774 rows=38106 loops=1)
                                    ->  Index Scan using i_p_type on part
(cost=0.00..4571.16 rows=1290 width=4) (actual time=0.215..33.542 rows=1266
loops=1)
                                          Index Cond: ((p_type)::text = 'PROMO
PLATED STEEL'::text)
                                    ->  Index Scan using i_l_partkey on lineitem
(cost=0.00..130.42 rows=32 width=20) (actual time=0.051..0.618 rows=30 loops=1266)
                                          Index Cond: ("outer".p_partkey =
lineitem.l_partkey)
                              ->  Hash  (cost=346.00..346.00 rows=10000
width=8) (actual time=24.783..24.783 rows=0 loops=1)
                                    ->  Seq Scan on supplier
(cost=0.00..346.00 rows=10000 width=8) (actual time=0.078..15.013 rows=10000
```

```
loops=1)
              ->  Hash  (cost=1.25..1.25 rows=25 width=33) (actual
time=0.092..0.092 rows=0 loops=1)
                    ->  Seq Scan on nation n2  (cost=0.00..1.25 rows=25 width=33)
(actual time=0.018..0.052 rows=25 loops=1)
 Total runtime: 3984.617 ms
(35 rows)
```

2.9 Q9

問い合わせをリスト 2.9-1 に示す。

リスト 2.9-1　Q9 の問い合わせ

```
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, supplier, 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 '%tan%' ) as profit group by nation, o_year order by
nation, o_year desc;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.9-2 に示す。

リスト 2.9-2　Q9 の実行計画

```
GroupAggregate  (cost=226206.05..226208.58 rows=92 width=49) (actual
time=33636.255..34600.041 rows=175 loops=1)
   ->  Sort  (cost=226206.05..226206.28 rows=92 width=49) (actual
time=33632.456..33899.188 rows=258959 loops=1)
         Sort Key: nation.n_name, date_part('year'::text,
(orders.o_orderdate)::timestamp without time zone)
         ->  Hash Join  (cost=1.31..226203.05 rows=92 width=49) (actual
time=6.578..28751.946 rows=258959 loops=1)
               Hash Cond: ("outer".s_nationkey = "inner".n_nationkey)
               ->  Nested Loop  (cost=0.00..226199.62 rows=94 width=24) (actual
time=6.457..27216.973 rows=258959 loops=1)
                     ->  Nested Loop  (cost=0.00..225918.20 rows=93 width=28)
```

```
(actual time=6.433..23393.288 rows=258959 loops=1)
                          ->  Nested Loop  (cost=0.00..225612.58 rows=101
width=28) (actual time=6.295..10223.859 rows=258959 loops=1)
                                ->  Nested Loop  (cost=0.00..32145.07 rows=32079
width=16) (actual time=0.118..1278.373 rows=34484 loops=1)
                                      ->  Seq Scan on part  (cost=0.00..7257.77
rows=8003 width=4) (actual time=0.043..630.503 rows=8621 loops=1)
                                            Filter: ((p_name)::text ~~
'%tan%'::text)
                                      ->  Index Scan using i_ps_partkey on
partsupp  (cost=0.00..3.06 rows=4 width=12) (actual time=0.032..0.054 rows=4
loops=8621)
                                            Index Cond: ("outer".p_partkey =
partsupp.ps_partkey)
                                ->  Index Scan using i_l_suppkey_partkey on
lineitem  (cost=0.00..6.02 rows=1 width=24) (actual time=0.098..0.234 rows=8
loops=34484)
                                      Index Cond: (("outer".p_partkey =
lineitem.l_partkey) AND ("outer".ps_suppkey = lineitem.l_suppkey))
                          ->  Index Scan using orders_pkey on orders
(cost=0.00..3.01 rows=1 width=8) (actual time=0.044..0.045 rows=1 loops=258959)
                                Index Cond: (orders.o_orderkey =
"outer".l_orderkey)
                    ->  Index Scan using supplier_pkey on supplier
(cost=0.00..3.01 rows=1 width=8) (actual time=0.008..0.010 rows=1 loops=258959)
                          Index Cond: ("outer".ps_suppkey = supplier.s_suppkey)
              ->  Hash  (cost=1.25..1.25 rows=25 width=33) (actual
time=0.069..0.069 rows=0 loops=1)
                    ->  Seq Scan on nation  (cost=0.00..1.25 rows=25 width=33)
(actual time=0.011..0.044 rows=25 loops=1)
 Total runtime: 34627.355 ms
(22 rows)
```

この問い合わせでは、(p_name like '%tan%')の検索がボトルネックになっており、改善の余地はない．これは、部分一致検索ではインデックスが使用されないからである。

PostgreSQL 8.0 での結果をリスト 2.9-3 に示す．これも改善の余地はなかった。

リスト 2.9-3　Q9 の実行計画

```
GroupAggregate  (cost=217990.20..217992.84 rows=96 width=49) (actual
time=28019.355..29031.965 rows=175 loops=1)
   ->  Sort  (cost=217990.20..217990.44 rows=96 width=49) (actual
time=28015.441..28282.198 rows=258943 loops=1)
         Sort Key: nation.n_name, date_part('year'::text,
(orders.o_orderdate)::timestamp without time zone)
         ->  Nested Loop  (cost=416.31..217987.04 rows=96 width=49) (actual
time=27.704..20960.745 rows=258943 loops=1)
               ->  Hash Join  (cost=416.31..217696.07 rows=96 width=49) (actual
time=27.615..8664.604 rows=258943 loops=1)
                     Hash Cond: ("outer".s_nationkey = "inner".n_nationkey)
                     ->  Nested Loop  (cost=415.00..217693.31 rows=96 width=24)
(actual time=27.463..7752.126 rows=258943 loops=1)
                           ->  Hash Join  (cost=415.00..37294.84 rows=29912
width=24) (actual time=27.379..1522.377 rows=34484 loops=1)
                                 Hash Cond: ("outer".ps_suppkey =
"inner".s_suppkey)
                                 ->  Nested Loop  (cost=0.00..35712.71 rows=32000
width=16) (actual time=0.156..1194.427 rows=34484 loops=1)
                                       ->  Seq Scan on part  (cost=0.00..7363.00
rows=8001 width=4) (actual time=0.060..614.311 rows=8621 loops=1)
                                             Filter: ((p_name)::text ~~
'%tan%'::text)
                                       ->  Index Scan using i_ps_partkey on
partsupp  (cost=0.00..3.28 rows=21 width=12) (actual time=0.033..0.048 rows=4
loops=8621)
                                             Index Cond: ("outer".p_partkey =
partsupp.ps_partkey)
                                 ->  Hash  (cost=346.00..346.00 rows=10000
width=8) (actual time=27.112..27.112 rows=0 loops=1)
                                       ->  Seq Scan on supplier
(cost=0.00..346.00 rows=10000 width=8) (actual time=0.029..16.270 rows=10000
loops=1)
                           ->  Index Scan using i_l_suppkey_partkey on lineitem
(cost=0.00..6.02 rows=1 width=24) (actual time=0.036..0.156 rows=8 loops=34484)
                                 Index Cond: (("outer".p_partkey =
lineitem.l_partkey) AND ("outer".ps_suppkey = lineitem.l_suppkey))
                     ->  Hash  (cost=1.25..1.25 rows=25 width=33) (actual
```

```
time=0.133..0.133 rows=0 loops=1)
                          ->  Seq Scan on nation  (cost=0.00..1.25 rows=25
width=33) (actual time=0.072..0.104 rows=25 loops=1)
               ->  Index Scan using orders_pkey on orders  (cost=0.00..3.01 rows=1
width=8) (actual time=0.038..0.040 rows=1 loops=258943)
                     Index Cond: (orders.o_orderkey = "outer".l_orderkey)
 Total runtime: 29055.020 ms
(23 rows)
```

2.10 Q10

問い合わせを図 1.2.10.1 に示す。

リスト 2.10-1 Q10 の問い合わせ

```
select c_custkey, c_name, sum(l_extendedprice * (1 - l_discount)) as revenue,
c_acctbal, n_name, c_address, c_phone, c_comment from customer, orders, lineitem,
nation where c_custkey = o_custkey and l_orderkey = o_orderkey and o_orderdate >=
date '1993-09-01' and o_orderdate < date '1993-09-01' + interval '3 month' and
l_returnflag = 'R' and c_nationkey = n_nationkey group by c_custkey, c_name,
c_acctbal, c_phone, n_name, c_address, c_comment order by revenue desc LIMIT 20;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.10-2 に示す。

リスト 2.10-2 Q10 の実行計画

```
Limit  (cost=595914.35..595914.40 rows=20 width=191) (actual
time=21071.346..21071.392 rows=20 loops=1)
   ->  Sort  (cost=595914.35..596783.48 rows=347652 width=191) (actual
time=21071.341..21071.361 rows=20 loops=1)
         Sort Key: sum((lineitem.l_extendedprice * (1::double precision -
lineitem.l_discount)))
         ->  GroupAggregate  (cost=465818.14..476247.70 rows=347652 width=191)
(actual time=19962.291..20596.091 rows=37732 loops=1)
               ->  Sort  (cost=465818.14..466687.27 rows=347652 width=191) (actual
time=19962.210..20117.950 rows=114094 loops=1)
                     Sort Key: customer.c_custkey, customer.c_name,
customer.c_acctbal, customer.c_phone, nation.n_name, customer.c_address,
```

```
customer.c_comment
                    ->  Hash Join  (cost=328414.37..346151.49 rows=347652
width=191) (actual time=17116.033..18764.464 rows=114094 loops=1)
                          Hash Cond: ("outer".c_nationkey = "inner".n_nationkey)
                          ->  Merge Join  (cost=328413.06..340066.30 rows=347650
width=166) (actual time=17115.921..18238.708 rows=114094 loops=1)
                                Merge Cond: ("outer".c_custkey =
"inner".o_custkey)
                                ->  Index Scan using customer_pkey on customer
(cost=0.00..6063.90 rows=150090 width=158) (actual time=0.084..351.756 rows=149997
loops=1)
                                ->  Sort  (cost=328413.06..329282.18 rows=347649
width=12) (actual time=17115.769..17244.958 rows=114094 loops=1)
                                      Sort Key: orders.o_custkey
                                      ->  Merge Join  (cost=0.00..290776.71
rows=347649 width=12) (actual time=0.675..16168.614 rows=114094 loops=1)
                                            Merge Cond: ("outer".o_orderkey =
"inner".l_orderkey)
                                            ->  Index Scan using orders_pkey on
orders  (cost=0.00..60858.28 rows=373335 width=8) (actual time=0.425..3090.068
rows=56601 loops=1)
                                                  Filter: ((o_orderdate >=
'1993-09-01'::date) AND ((o_orderdate)::timestamp without time zone < '1993-12-01
00:00:00'::timestamp without time zone))
                                            ->  Index Scan using i_l_orderkey on
lineitem  (cost=0.00..221707.08 rows=1523061 width=12) (actual
time=0.109..11172.256 rows=1478792 loops=1)
                                                  Filter: (l_returnflag =
'R'::bpchar)
                          ->  Hash  (cost=1.25..1.25 rows=25 width=33) (actual
time=0.087..0.087 rows=0 loops=1)
                                ->  Seq Scan on nation  (cost=0.00..1.25 rows=25
width=33) (actual time=0.028..0.060 rows=25 loops=1)
 Total runtime: 21207.822 ms
(22 rows)
```

sort_mem を増やし(set sort_mem to 102400)、merge join を hash join にすることで高速化できた(リスト 2.10-3)。

リスト 2.10-3　sort_mem を増やした後の実行計画

```
Limit  (cost=415004.01..415004.06 rows=20 width=191) (actual
time=13436.845..13436.880 rows=20 loops=1)
   ->  Sort  (cost=415004.01..415965.13 rows=384446 width=191) (actual
time=13436.841..13436.853 rows=20 loops=1)
         Sort Key: sum((lineitem.l_extendedprice * (1::double precision -
lineitem.l_discount)))
         ->  GroupAggregate  (cost=367808.61..379341.99 rows=384446 width=191)
(actual time=12647.188..13224.105 rows=37760 loops=1)
               ->  Sort  (cost=367808.61..368769.73 rows=384446 width=191) (actual
time=12647.138..12722.190 rows=114232 loops=1)
                     Sort Key: customer.c_custkey, customer.c_name,
customer.c_acctbal, customer.c_phone, nation.n_name, customer.c_address,
customer.c_comment
                     ->  Hash Join  (cost=313342.80..332146.59 rows=384446
width=191) (actual time=10473.698..12074.327 rows=114232 loops=1)
                           Hash Cond: ("outer".c_nationkey = "inner".n_nationkey)
                           ->  Merge Join  (cost=313341.49..325417.53 rows=384442
width=166) (actual time=10473.280..11571.618 rows=114232 loops=1)
                                 Merge Cond: ("outer".c_custkey =
"inner".o_custkey)
                                 ->  Index Scan using customer_pkey on customer
(cost=0.00..5935.00 rows=150000 width=158) (actual time=0.104..342.270 rows=149997
loops=1)
                                 ->  Sort  (cost=313341.49..314302.60 rows=384442
width=12) (actual time=10473.109..10548.968 rows=114232 loops=1)
                                       Sort Key: orders.o_custkey
                                       ->  Hash Join  (cost=55776.52..277679.87
rows=384442 width=12) (actual time=1976.213..10182.716 rows=114232 loops=1)
                                             Hash Cond: ("outer".l_orderkey =
"inner".o_orderkey)
                                             ->  Seq Scan on lineitem
(cost=0.00..202840.74 rows=1521819 width=12) (actual time=0.066..6548.729
rows=1480362 loops=1)
                                                   Filter: (l_returnflag =
'R'::bpchar)
                                             ->  Hash  (cost=54828.25..54828.25
rows=379309 width=8) (actual time=1975.008..1975.008 rows=0 loops=1)
                                                   ->  Seq Scan on orders
```

```
(cost=0.00..54828.25 rows=379309 width=8) (actual time=0.067..1897.219 rows=56654
loops=1)
                                                    Filter: ((o_orderdate >=
'1993-09-01'::date) AND ((o_orderdate)::timestamp without time zone < '1993-12-01
00:00:00'::timestamp without time zone))
                          ->  Hash  (cost=1.25..1.25 rows=25 width=33) (actual
time=0.098..0.098 rows=0 loops=1)
                                ->  Seq Scan on nation  (cost=0.00..1.25 rows=25
width=33) (actual time=0.010..0.042 rows=25 loops=1)
 Total runtime: 13509.973 ms
(23 rows)
```

この問い合わせでは、使用されていないインデックスがある。それらを使用させるために、postgresql.conf で以下の設定を行う。

```
effective_cache_size = 100000;
random_page_cost = 2;
```

その結果、lineitem の l_orderkey に設定したインデックスが使われるようになり、性能が改善された(リスト 2.10-4)。

リスト 2.10-4　インデックスが使用されるようになり、改善された問い合わせ計画

```
Limit  (cost=242173.17..242173.22 rows=20 width=191) (actual
time=7601.896..7601.933 rows=20 loops=1)
   ->  Sort  (cost=242173.17..242307.74 rows=53827 width=191) (actual
time=7601.891..7601.904 rows=20 loops=1)
         Sort Key: sum((lineitem.l_extendedprice * (1::double precision -
lineitem.l_discount)))
         ->  HashAggregate  (cost=237539.73..237943.44 rows=53827 width=191)
(actual time=7263.341..7397.863 rows=37760 loops=1)
               ->  Nested Loop  (cost=55243.81..236463.19 rows=53827 width=191)
(actual time=1842.933..6540.445 rows=114232 loops=1)
                     ->  Hash Join  (cost=55243.81..63279.21 rows=53108 width=187)
(actual time=1842.771..2966.616 rows=56654 loops=1)
                           Hash Cond: ("outer".c_nationkey = "inner".n_nationkey)
                           ->  Merge Join  (cost=55242.50..62348.52 rows=53107
width=162) (actual time=1841.889..2720.722 rows=56654 loops=1)
                                 Merge Cond: ("outer".c_custkey =
```

```
"inner".o_custkey)
                                ->  Index Scan using customer_pkey on customer
(cost=0.00..5935.00 rows=150000 width=158) (actual time=0.113..369.613 rows=150000
loops=1)
                                ->  Sort  (cost=55242.50..55375.27 rows=53107
width=8) (actual time=1841.709..1885.987 rows=56654 loops=1)
                                      Sort Key: orders.o_custkey
                                      ->  Seq Scan on orders
(cost=0.00..51074.50 rows=53107 width=8) (actual time=0.042..1692.528 rows=56654
loops=1)
                                            Filter: ((o_orderdate >=
'1993-09-01'::date) AND (o_orderdate < '1993-12-01'::date))
                           ->  Hash  (cost=1.25..1.25 rows=25 width=33) (actual
time=0.164..0.164 rows=0 loops=1)
                                ->  Seq Scan on nation  (cost=0.00..1.25 rows=25
width=33) (actual time=0.086..0.120 rows=25 loops=1)
                     ->  Index Scan using i_l_orderkey on lineitem
(cost=0.00..3.24 rows=2 width=12) (actual time=0.045..0.053 rows=2 loops=56654)
                           Index Cond: (lineitem.l_orderkey = "outer".o_orderkey)
                           Filter: (l_returnflag = 'R'::bpchar)
 Total runtime: 7634.513 ms
(20 rows)
```

PostgreSQL 8.0 での問い合わせ実行計画をリスト 2.10-5 に示す。

リスト 2.10-5　PostgreSQL8.0 での実行計画

```
Limit  (cost=343739.26..343739.31 rows=20 width=191) (actual
time=19131.560..19131.607 rows=20 loops=1)
   ->  Sort  (cost=343739.26..343904.62 rows=66141 width=191) (actual
time=19131.555..19131.575 rows=20 loops=1)
         Sort Key: sum((lineitem.l_extendedprice * (1::double precision -
lineitem.l_discount)))
         ->  GroupAggregate  (cost=321689.37..323673.60 rows=66141 width=191)
(actual time=17452.742..18660.977 rows=37728 loops=1)
               ->  Sort  (cost=321689.37..321854.72 rows=66141 width=191) (actual
time=17452.644..17612.576 rows=114106 loops=1)
                     Sort Key: customer.c_custkey, customer.c_name,
customer.c_acctbal, customer.c_phone, nation.n_name, customer.c_address,
```

```
customer.c_comment
                    ->  Hash Join  (cost=293127.33..301623.70 rows=66141
width=191) (actual time=13856.062..15309.717 rows=114106 loops=1)
                          Hash Cond: ("outer".c_nationkey = "inner".n_nationkey)
                          ->  Merge Join  (cost=293126.02..300630.28 rows=66141
width=166) (actual time=13855.943..14853.344 rows=114106 loops=1)
                                Merge Cond: ("outer".c_custkey =
"inner".o_custkey)
                                ->  Index Scan using customer_pkey on customer
(cost=0.00..6138.01 rows=150001 width=158) (actual time=0.082..284.572 rows=149997
loops=1)
                                ->  Sort  (cost=293126.02..293291.37 rows=66141
width=12) (actual time=13855.797..13986.528 rows=114106 loops=1)
                                      Sort Key: orders.o_custkey
                                      ->  Merge Join  (cost=0.00..287133.81
rows=66141 width=12) (actual time=0.234..12940.122 rows=114106 loops=1)
                                            Merge Cond: ("outer".o_orderkey =
"inner".l_orderkey)
                                            ->  Index Scan using orders_pkey on
orders  (cost=0.00..57852.26 rows=66031 width=8) (actual time=0.111..2060.358
rows=56592 loops=1)
                                                  Filter: ((o_orderdate >=
'1993-09-01'::date) AND (o_orderdate < '1993-12-01 00:00:00'::timestamp without time
zone))
                                            ->  Index Scan using i_l_orderkey on
lineitem  (cost=0.00..224756.40 rows=1502305 width=12) (actual
time=0.110..9181.804 rows=1478854 loops=1)
                                                  Filter: (l_returnflag =
'R'::bpchar)
                          ->  Hash  (cost=1.25..1.25 rows=25 width=33) (actual
time=0.094..0.094 rows=0 loops=1)
                                ->  Seq Scan on nation  (cost=0.00..1.25 rows=25
width=33) (actual time=0.031..0.067 rows=25 loops=1)
 Total runtime: 19176.247 ms
(22 rows)
```

7.4 同様、ソートメモリを増やして若干の改善が見られた(リスト 2.10-6)。

リスト 2.10-6　8.0 でソートメモリを増やした場合の実行計画

```
Limit  (cost=286623.63..286623.68 rows=20 width=191) (actual
time=15279.674..15279.706 rows=20 loops=1)
   ->  Sort  (cost=286623.63..286788.98 rows=66141 width=191) (actual
time=15279.668..15279.678 rows=20 loops=1)
         Sort Key: sum((lineitem.l_extendedprice * (1::double precision -
lineitem.l_discount)))
         ->  HashAggregate  (cost=280831.91..281327.96 rows=66141 width=191)
(actual time=14936.081..15073.013 rows=37728 loops=1)
               ->  Hash Join  (cost=65722.96..279509.09 rows=66141 width=191)
(actual time=2602.078..13640.469 rows=114106 loops=1)
                     Hash Cond: ("outer".l_orderkey = "inner".o_orderkey)
                     ->  Seq Scan on lineitem  (cost=0.00..205613.19 rows=1502305
width=12) (actual time=0.061..6678.903 rows=1478870 loops=1)
                           Filter: (l_returnflag = 'R'::bpchar)
                     ->  Hash  (cost=65557.89..65557.89 rows=66031 width=187)
(actual time=2601.804..2601.804 rows=0 loops=1)
                           ->  Hash Join  (cost=57064.81..65557.89 rows=66031
width=187) (actual time=1635.446..2497.816 rows=56592 loops=1)
                                 Hash Cond: ("outer".c_nationkey =
"inner".n_nationkey)
                                 ->  Merge Join  (cost=57063.50..64566.11
rows=66031 width=162) (actual time=1634.641..2291.275 rows=56592 loops=1)
                                       Merge Cond: ("outer".c_custkey =
"inner".o_custkey)
                                       ->  Index Scan using customer_pkey on
customer  (cost=0.00..6138.01 rows=150001 width=158) (actual time=0.106..245.179
rows=150000 loops=1)
                                       ->  Sort  (cost=57063.50..57228.58
rows=66031 width=8) (actual time=1634.475..1672.238 rows=56592 loops=1)
                                             Sort Key: orders.o_custkey
                                             ->  Seq Scan on orders
(cost=0.00..51777.43 rows=66031 width=8) (actual time=0.044..1489.770 rows=56592
loops=1)
                                                   Filter: ((o_orderdate >=
'1993-09-01'::date) AND (o_orderdate < '1993-12-01 00:00:00'::timestamp without time
zone))
                                 ->  Hash  (cost=1.25..1.25 rows=25 width=33)
(actual time=0.149..0.149 rows=0 loops=1)
```

```
                                        ->  Seq Scan on nation  (cost=0.00..1.25
rows=25 width=33) (actual time=0.076..0.108 rows=25 loops=1)
 Total runtime: 15320.225 ms
(21 rows)
```

8.0 では effective_cache_size = 100000、random_page_cost = 2 をしても lineitem の l_orderkey のインデックスが使われることはなく、これ以上の改善はなかった。

2.11 Q11

問い合わせをリスト 2.11-1 に示す。

リスト 2.11-1 Q11 の問い合わせ

```
select ps_partkey, sum(ps_supplycost * ps_availqty) as value from partsupp, supplier,
nation where ps_suppkey = s_suppkey and s_nationkey = n_nationkey and n_name = 'KENYA'
group by ps_partkey having sum(ps_supplycost * ps_availqty) > ( select
sum(ps_supplycost * ps_availqty) * 0.0001000000 from partsupp, supplier, nation where
ps_suppkey = s_suppkey and s_nationkey = n_nationkey and n_name = 'KENYA' ) order
by value desc;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.11-2 に示す。

リスト 2.11-2 Q11 の実行計画

```
Sort  (cost=74540.63..74701.20 rows=64231 width=12) (actual
time=3784.268..3785.020 rows=1217 loops=1)
   Sort Key: sum((partsupp.ps_supplycost * (partsupp.ps_availqty)::double
precision))
   InitPlan
     ->  Aggregate  (cost=30749.60..30749.60 rows=1 width=8) (actual
time=1825.435..1825.435 rows=1 loops=1)
           ->  Hash Join  (cost=402.94..30589.02 rows=64231 width=8) (actual
time=22.222..1776.826 rows=30080 loops=1)
                 Hash Cond: ("outer".ps_suppkey = "inner".s_suppkey)
                 ->  Seq Scan on partsupp  (cost=0.00..25534.18 rows=801918
width=12) (actual time=0.024..1008.643 rows=800000 loops=1)
                 ->  Hash  (cost=400.93..400.93 rows=804 width=4) (actual
```

```
time=22.138..22.138 rows=0 loops=1)
                      ->  Hash Join  (cost=1.32..400.93 rows=804 width=4) (actual
time=0.147..21.641 rows=376 loops=1)
                            Hash Cond: ("outer".s_nationkey =
"inner".n_nationkey)
                            ->  Seq Scan on supplier  (cost=0.00..341.38
rows=10038 width=8) (actual time=0.029..12.990 rows=10000 loops=1)
                            ->  Hash  (cost=1.31..1.31 rows=2 width=4) (actual
time=0.082..0.082 rows=0 loops=1)
                                  ->  Seq Scan on nation  (cost=0.00..1.31 rows=2
width=4) (actual time=0.062..0.070 rows=1 loops=1)
                                        Filter: (n_name = 'KENYA'::bpchar)
   ->  GroupAggregate  (cost=36306.84..38073.20 rows=64231 width=12) (actual
time=3692.170..3781.814 rows=1217 loops=1)
         Filter: (sum((ps_supplycost * (ps_availqty)::double precision)) > $0)
         ->  Sort  (cost=36306.84..36467.42 rows=64231 width=12) (actual
time=1866.047..1894.107 rows=30080 loops=1)
               Sort Key: partsupp.ps_partkey
               ->  Hash Join  (cost=402.94..30589.02 rows=64231 width=12) (actual
time=22.058..1761.196 rows=30080 loops=1)
                     Hash Cond: ("outer".ps_suppkey = "inner".s_suppkey)
                     ->  Seq Scan on partsupp  (cost=0.00..25534.18 rows=801918
width=16) (actual time=0.045..1022.041 rows=800000 loops=1)
                     ->  Hash  (cost=400.93..400.93 rows=804 width=4) (actual
time=21.976..21.976 rows=0 loops=1)
                           ->  Hash Join  (cost=1.32..400.93 rows=804 width=4)
(actual time=0.113..21.578 rows=376 loops=1)
                                 Hash Cond: ("outer".s_nationkey =
"inner".n_nationkey)
                                 ->  Seq Scan on supplier  (cost=0.00..341.38
rows=10038 width=8) (actual time=0.023..12.922 rows=10000 loops=1)
                                 ->  Hash  (cost=1.31..1.31 rows=2 width=4)
(actual time=0.068..0.068 rows=0 loops=1)
                                       ->  Seq Scan on nation  (cost=0.00..1.31
rows=2 width=4) (actual time=0.055..0.062 rows=1 loops=1)
                                             Filter: (n_name = 'KENYA'::bpchar)
 Total runtime: 3787.441 ms
(29 rows)
```

特に改善の余地は見いだせなかった。

PostgreSQL 8.0 での実行結果をリスト 2.11-3 に示す。

リスト 2.11-3　PostgreSQL8.0 での実行計画

```
Sort  (cost=67173.28..67248.05 rows=29909 width=12) (actual
time=3772.806..3773.550 rows=1217 loops=1)
   Sort Key: sum((partsupp.ps_supplycost * (partsupp.ps_availqty)::double
precision))
   InitPlan
     ->  Aggregate  (cost=30690.53..30690.54 rows=1 width=8) (actual
time=1803.080..1803.081 rows=1 loops=1)
           ->  Hash Join  (cost=402.31..30615.75 rows=29909 width=8) (actual
time=22.357..1761.016 rows=30080 loops=1)
                 Hash Cond: ("outer".ps_suppkey = "inner".s_suppkey)
                 ->  Seq Scan on partsupp  (cost=0.00..25914.90 rows=799890
width=12) (actual time=0.025..1058.980 rows=800000 loops=1)
                 ->  Hash  (cost=401.31..401.31 rows=400 width=4) (actual
time=22.272..22.272 rows=0 loops=1)
                       ->  Hash Join  (cost=1.31..401.31 rows=400 width=4) (actual
time=0.157..21.805 rows=376 loops=1)
                             Hash Cond: ("outer".s_nationkey =
"inner".n_nationkey)
                             ->  Seq Scan on supplier  (cost=0.00..346.00
rows=10000 width=8) (actual time=0.028..13.460 rows=10000 loops=1)
                             ->  Hash  (cost=1.31..1.31 rows=1 width=4) (actual
time=0.091..0.091 rows=0 loops=1)
                                   ->  Seq Scan on nation  (cost=0.00..1.31 rows=1
width=4) (actual time=0.071..0.079 rows=1 loops=1)
                                         Filter: (n_name = 'KENYA'::bpchar)
   ->  GroupAggregate  (cost=33138.00..33960.50 rows=29909 width=12) (actual
time=3687.119..3770.420 rows=1217 loops=1)
         Filter: (sum((ps_supplycost * (ps_availqty)::double precision)) > $0)
         ->  Sort  (cost=33138.00..33212.77 rows=29909 width=12) (actual
time=1883.396..1911.416 rows=30080 loops=1)
               Sort Key: partsupp.ps_partkey
               ->  Hash Join  (cost=402.31..30615.75 rows=29909 width=12) (actual
time=22.578..1777.981 rows=30080 loops=1)
                     Hash Cond: ("outer".ps_suppkey = "inner".s_suppkey)
                     ->  Seq Scan on partsupp  (cost=0.00..25914.90 rows=799890
```

```
width=16) (actual time=0.042..1068.653 rows=800000 loops=1)
                    ->  Hash  (cost=401.31..401.31 rows=400 width=4) (actual
time=22.498..22.498 rows=0 loops=1)
                          ->  Hash Join  (cost=1.31..401.31 rows=400 width=4)
(actual time=0.132..22.118 rows=376 loops=1)
                                Hash Cond: ("outer".s_nationkey =
"inner".n_nationkey)
                                ->  Seq Scan on supplier  (cost=0.00..346.00
rows=10000 width=8) (actual time=0.023..13.651 rows=10000 loops=1)
                                ->  Hash  (cost=1.31..1.31 rows=1 width=4)
(actual time=0.085..0.085 rows=0 loops=1)
                                      ->  Seq Scan on nation  (cost=0.00..1.31
rows=1 width=4) (actual time=0.074..0.082 rows=1 loops=1)
                                            Filter: (n_name = 'KENYA'::bpchar)
 Total runtime: 3776.399 ms
(29 rows)
```

7.4 同様、改善の余地はなかった。

2.12 Q12

問い合わせをリスト 2.12-1 に示す。

リスト 2.12-1 Q12 の問い合わせ

```
select l_shipmode, sum(case when o_orderpriority = '1-URGENT' or o_orderpriority =
'2-HIGH' then 1 else 0 end) as high_line_count, sum(case when o_orderpriority <>
'1-URGENT' and o_orderpriority <> '2-HIGH' then 1 else 0 end) as low_line_count from
orders, lineitem where o_orderkey = l_orderkey and l_shipmode in ('TRUCK', 'RAIL')
and l_commitdate < l_receiptdate and l_shipdate < l_commitdate and l_receiptdate >=
date '1997-01-01' and l_receiptdate < date '1997-01-01' + interval '1 year' group
by l_shipmode order by l_shipmode;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.12-2 に示す。

リスト 2.12-2 Q12 の実行計画

```
Sort  (cost=337478.70..337478.70 rows=1 width=33) (actual
```

```
time=11505.028..11505.030 rows=2 loops=1)
   Sort Key: lineitem.l_shipmode
   ->  HashAggregate  (cost=337478.67..337478.69 rows=1 width=33) (actual
time=11505.009..11505.011 rows=2 loops=1)
         ->  Nested Loop  (cost=0.00..337376.70 rows=13596 width=33) (actual
time=11.775..11344.915 rows=31209 loops=1)
               ->  Seq Scan on lineitem  (cost=0.00..292522.30 rows=14823 width=18)
(actual time=11.630..10451.711 rows=31209 loops=1)
                     Filter: (((l_shipmode = 'TRUCK'::bpchar) OR (l_shipmode =
'RAIL'::bpchar)) AND (l_commitdate < l_receiptdate) AND (l_shipdate < l_commitdate)
AND (l_receiptdate >= '1997-01-01'::date) AND ((l_receiptdate)::timestamp without
time zone < '1998-01-01 00:00:00'::timestamp without time zone))
               ->  Index Scan using orders_pkey on orders  (cost=0.00..3.01 rows=1
width=23) (actual time=0.021..0.023 rows=1 loops=31209)
                     Index Cond: (orders.o_orderkey = "outer".l_orderkey)
 Total runtime: 11505.304 ms
(9 rows)
```

特に改善の余地は見いだせなかった。

PostgreSQL 8.0 での実行結果をリスト 2.12-3 に示す。

リスト 2.12-3　PostgreSQL8.0 での実行計画

```
Sort  (cost=337656.80..337656.80 rows=1 width=33) (actual
time=13940.988..13940.990 rows=2 loops=1)
   Sort Key: lineitem.l_shipmode
   ->  HashAggregate  (cost=337656.77..337656.79 rows=1 width=33) (actual
time=13940.972..13940.975 rows=2 loops=1)
         ->  Merge Join  (cost=282988.35..337450.35 rows=27523 width=33) (actual
time=10221.808..13805.268 rows=31193 loops=1)
               Merge Cond: ("outer".o_orderkey = "inner".l_orderkey)
               ->  Index Scan using orders_pkey on orders  (cost=0.00..50353.12
rows=1499829 width=23) (actual time=0.122..2157.721 rows=1499977 loops=1)
               ->  Sort  (cost=282988.35..283057.15 rows=27523 width=18) (actual
time=10221.326..10258.389 rows=31193 loops=1)
                     Sort Key: lineitem.l_orderkey
                     ->  Seq Scan on lineitem  (cost=0.00..280628.38 rows=27523
width=18) (actual time=0.865..10032.263 rows=31193 loops=1)
                           Filter: (((l_shipmode = 'TRUCK'::bpchar) OR (l_shipmode
```

```
= 'RAIL'::bpchar)) AND (l_commitdate < l_receiptdate) AND (l_shipdate < l_commitdate)
AND (l_receiptdate >= '1997-01-01'::date) AND (l_receiptdate < '1998-01-01
00:00:00'::timestamp without time zone))
 Total runtime: 13943.080 ms
(11 rows)
```

7.4 同様、改善の余地はなかった。

2.13 Q13

問い合わせをリスト 2.13-1 に示す。

リスト 2.13-1 Q13 の問い合わせ

```
select c_count, count(*) as custdist from ( select c_custkey, count(o_orderkey) from
customer left outer join orders on c_custkey = o_custkey and o_comment not like
'%pending%deposits%' group by c_custkey ) as c_orders (c_custkey, c_count) group by
c_count order by custdist desc, c_count desc;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.13-2 に示す。

リスト 2.13-2 Q13 の実行計画

```
Sort  (cost=294844.59..294845.09 rows=200 width=8) (actual
time=27114.365..27114.390 rows=41 loops=1)
   Sort Key: count(*), c_count
   ->  HashAggregate  (cost=294836.45..294836.95 rows=200 width=8) (actual
time=27114.244..27114.309 rows=41 loops=1)
         ->  Subquery Scan c_orders  (cost=255770.88..294086.00 rows=150090
width=8) (actual time=19984.958..26907.071 rows=150000 loops=1)
               ->  GroupAggregate  (cost=255770.88..292585.10 rows=150090
width=8) (actual time=19984.953..26648.820 rows=150000 loops=1)
                     ->  Merge Left Join  (cost=255770.88..284709.22 rows=1500131
width=8) (actual time=19984.854..24835.044 rows=1529967 loops=1)
                           Merge Cond: ("outer".c_custkey = "inner".o_custkey)
                           ->  Index Scan using customer_pkey on customer
(cost=0.00..6063.90 rows=150090 width=4) (actual time=0.080..359.309 rows=150000
loops=1)
```

```
                        ->  Sort  (cost=255770.88..259521.21 rows=1500130
width=8) (actual time=19984.744..21400.849 rows=1479963 loops=1)
                              Sort Key: orders.o_custkey
                              ->  Seq Scan on orders  (cost=0.00..47282.62
rows=1500130 width=8) (actual time=0.194..7381.567 rows=1479963 loops=1)
                                    Filter: ((o_comment)::text !~~
'%pending%deposits%'::text)
 Total runtime: 27166.965 ms
(13 rows)
```

特に改善の余地は見いだせなかった。

PostgreSQL 8.0 での実行結果をリスト 2.13-3 に示す。

リスト 2.13-3　PostgreSQL8.0 での実行計画

```
Sort  (cost=303944.96..303945.46 rows=200 width=8) (actual
time=26500.888..26500.915 rows=41 loops=1)
   Sort Key: count(*), c_count
   ->  HashAggregate  (cost=303936.81..303937.31 rows=200 width=8) (actual
time=26500.736..26500.782 rows=41 loops=1)
         ->  Subquery Scan c_orders  (cost=264803.07..303186.81 rows=150001
width=8) (actual time=19772.026..26309.295 rows=150000 loops=1)
               ->  GroupAggregate  (cost=264803.07..301686.80 rows=150001
width=8) (actual time=19772.021..26056.829 rows=150000 loops=1)
                     ->  Merge Left Join  (cost=264803.07..293812.65 rows=1499829
width=8) (actual time=19771.934..24317.965 rows=1529940 loops=1)
                           Merge Cond: ("outer".c_custkey = "inner".o_custkey)
                           ->  Index Scan using customer_pkey on customer
(cost=0.00..6138.01 rows=150001 width=4) (actual time=0.102..265.126 rows=150000
loops=1)
                           ->  Sort  (cost=264803.07..268552.64 rows=1499829
width=8) (actual time=19771.803..21221.748 rows=1479936 loops=1)
                                 Sort Key: orders.o_custkey
                                 ->  Seq Scan on orders  (cost=0.00..48027.86
rows=1499829 width=8) (actual time=0.030..7203.678 rows=1479936 loops=1)
                                       Filter: ((o_comment)::text !~~
'%pending%deposits%'::text)
 Total runtime: 26558.664 ms
(13 rows)
```

7.4 同様、改善の余地はなかった。

2.14 Q14

問い合わせを図 1.2.14.1 に示す。

リスト 2.14-1 Q14 の問い合わせ

```
select 100.00 * sum(case when p_type like 'PROMO%' then l_extendedprice * (1 -
l_discount) else 0 end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from lineitem, part where l_partkey = p_partkey and l_shipdate >= date '1997-04-01'
and l_shipdate < date '1997-04-01' + interval '1 month';
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.14-2 に示す。

リスト 2.14-2 Q14 の実行計画

```
Aggregate  (cost=258998.27..258998.29 rows=1 width=32) (actual
time=8506.586..8506.587 rows=1 loops=1)
   ->  Hash Join  (cost=8527.77..256853.39 rows=428974 width=32) (actual
time=717.232..8322.719 rows=74553 loops=1)
         Hash Cond: ("outer".l_partkey = "inner".p_partkey)
         ->  Seq Scan on lineitem  (cost=0.00..232559.29 rows=428973 width=12)
(actual time=45.120..6653.133 rows=74553 loops=1)
               Filter: ((l_shipdate >= '1997-04-01'::date) AND
((l_shipdate)::timestamp without time zone < '1997-05-01 00:00:00'::timestamp
without time zone))
         ->  Hash  (cost=6757.62..6757.62 rows=200062 width=28) (actual
time=667.440..667.440 rows=0 loops=1)
               ->  Seq Scan on part  (cost=0.00..6757.62 rows=200062 width=28)
(actual time=0.046..351.469 rows=200000 loops=1)
 Total runtime: 8507.199 ms
(8 rows)
```

この問い合わせ計画では、lineitem の l_shipdate 列のインデックスが使用されていない。そこで、「date '1997-04-01' + interval '1 month';」を「(date '1997-04-01' + interval '1 month')::date;」に書き換え、更に postgresql.conf の random_page_cost を 3 にして

改善があった(リスト 2.14-3)。

リスト 2.14-3　改善後の実行計画

```
Aggregate  (cost=183069.92..183069.94 rows=1 width=32) (actual
time=3151.930..3151.931 rows=1 loops=1)
   ->  Merge Join  (cost=173857.57..182749.49 rows=64085 width=32) (actual
time=2011.774..2964.871 rows=74626 loops=1)
         Merge Cond: ("outer".p_partkey = "inner".l_partkey)
         ->  Index Scan using part_pkey on part  (cost=0.00..7398.00 rows=200000
width=28) (actual time=0.103..436.866 rows=200000 loops=1)
         ->  Sort  (cost=173857.57..174034.48 rows=70765 width=12) (actual
time=2011.614..2090.585 rows=74626 loops=1)
               Sort Key: lineitem.l_partkey
               ->  Index Scan using i_l_shipdate on lineitem  (cost=0.00..167535.18
rows=70765 width=12) (actual time=0.108..1447.932 rows=74626 loops=1)
                     Index Cond: ((l_shipdate >= '1997-04-01'::date) AND
(l_shipdate < '1997-05-01'::date))
 Total runtime: 3155.129 ms
(9 rows)
```

PostgreSQL 8.0 での実行結果をリスト 2.14-4 に示す。

リスト 2.14-4　PostgreSQL8.0 での実行結果

```
Aggregate  (cost=236613.93..236613.95 rows=1 width=32) (actual
time=6861.079..6861.080 rows=1 loops=1)
   ->  Merge Join  (cost=227036.89..236261.74 rows=70437 width=32) (actual
time=5809.466..6689.221 rows=74537 loops=1)
         Merge Cond: ("outer".p_partkey = "inner".l_partkey)
         ->  Index Scan using part_pkey on part  (cost=0.00..7675.00 rows=200000
width=28) (actual time=8.208..401.618 rows=200000 loops=1)
         ->  Sort  (cost=227036.89..227212.98 rows=70437 width=12) (actual
time=5801.202..5876.573 rows=74537 loops=1)
               Sort Key: lineitem.l_partkey
               ->  Seq Scan on lineitem  (cost=0.00..220616.23 rows=70437 width=12)
(actual time=0.083..5247.125 rows=74537 loops=1)
                     Filter: ((l_shipdate >= '1997-04-01'::date) AND (l_shipdate <
'1997-05-01 00:00:00'::timestamp without time zone))
```

```
 Total runtime: 6865.324 ms
(9 rows)
```

8.0 でも lineitem の l_shipdate 列のインデックスが使用されていないため、set random_page_cost to 3 にして 7.4 同様に改善が見られた(リスト 2.14-5)。

リスト 2.14-5　8.0 での改善後の問い合わせ計画

```
Aggregate  (cost=226909.77..226909.79 rows=1 width=32) (actual
time=2415.255..2415.255 rows=1 loops=1)
   ->  Merge Join  (cost=217332.73..226557.58 rows=70437 width=32) (actual
time=1463.543..2247.095 rows=74537 loops=1)
         Merge Cond: ("outer".p_partkey = "inner".l_partkey)
         ->  Index Scan using part_pkey on part  (cost=0.00..7675.00 rows=200000
width=28) (actual time=0.134..306.318 rows=200000 loops=1)
         ->  Sort  (cost=217332.73..217508.82 rows=70437 width=12) (actual
time=1463.355..1538.286 rows=74537 loops=1)
               Sort Key: lineitem.l_partkey
               ->  Index Scan using i_l_shipdate on lineitem  (cost=0.00..210932.42
rows=70437 width=12) (actual time=2.905..922.830 rows=74537 loops=1)
                     Index Cond: ((l_shipdate >= '1997-04-01'::date) AND
(l_shipdate < '1997-05-01 00:00:00'::timestamp without time zone))
 Total runtime: 2419.472 ms
(9 rows)
```

2.15　Q15

問い合わせをリスト 2.15-1 に示す。

リスト 2.15-1　Q15 の問い合わせ

```
create view revenue0 (supplier_no, total_revenue)
as select l_suppkey, sum(l_extendedprice * (1 - l_discount))
from lineitem where l_shipdate >= '1997-02-01'
and l_shipdate < date'1997-02-01' + interval '90 days'
group by l_suppkey;

select s_suppkey, s_name, s_address, s_phone, total_revenue from supplier, revenue0
where s_suppkey = supplier_no and total_revenue = (select max(total_revenue) from
```

```
revenue0 ) order by s_suppkey;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.15-2 に示す。

リスト 2.15-2　Q15 の実行計画

```
Merge Join  (cost=472151.89..472564.25 rows=824 width=88) (actual
time=15308.754..15308.761 rows=1 loops=1)
   Merge Cond: ("outer".s_suppkey = "inner".supplier_no)
   InitPlan
     ->  Aggregate  (cost=235420.96..235420.96 rows=1 width=8) (actual
time=7556.644..7556.645 rows=1 loops=1)
           ->  Subquery Scan revenue0  (cost=235404.50..235418.90 rows=823
width=8) (actual time=7514.550..7547.205 rows=10000 loops=1)
                 ->  HashAggregate  (cost=235404.50..235410.67 rows=823 width=12)
(actual time=7514.545..7531.924 rows=10000 loops=1)
                       ->  Seq Scan on lineitem  (cost=0.00..232876.63 rows=505574
width=12) (actual time=0.119..6865.191 rows=225048 loops=1)
                             Filter: ((l_shipdate >= '1997-02-01'::date) AND
((l_shipdate)::timestamp without time zone < '1997-05-02 00:00:00'::timestamp
without time zone))
   ->  Index Scan using supplier_pkey on supplier  (cost=0.00..375.00 rows=10000
width=80) (actual time=0.116..2.043 rows=966 loops=1)
   ->  Sort  (cost=236730.92..236732.98 rows=823 width=12) (actual
time=15305.863..15305.864 rows=1 loops=1)
         Sort Key: revenue0.supplier_no
         ->  Subquery Scan revenue0  (cost=236668.44..236691.07 rows=823 width=12)
(actual time=15302.953..15305.849 rows=1 loops=1)
               ->  HashAggregate  (cost=236668.44..236682.84 rows=823 width=12)
(actual time=15302.947..15305.838 rows=1 loops=1)
                     Filter: (sum((l_extendedprice * (1::double precision -
l_discount))) = $0)
                     ->  Seq Scan on lineitem  (cost=0.00..232876.63 rows=505574
width=12) (actual time=0.091..7095.275 rows=225048 loops=1)
                           Filter: ((l_shipdate >= '1997-02-01'::date) AND
((l_shipdate)::timestamp without time zone < '1997-05-02 00:00:00'::timestamp
without time zone))
 Total runtime: 15310.548 ms
```

```
(17 rows)
```

特に改善する余地はなかった。

PostgreSQL 8.0 の実行計画をリスト 2.15-3 に示す。7.4 との大きな違いはない。こちらも特に改善の余地はなかった。

リスト 2.15-3　PostgreSQL8.0 の実行計画

```
Merge Join  (cost=443903.02..444320.92 rows=327 width=89) (actual
time=12126.841..12126.847 rows=1 loops=1)
   Merge Cond: ("outer".s_suppkey = "inner".supplier_no)
   InitPlan
     ->  Aggregate  (cost=221679.32..221679.32 rows=1 width=8) (actual
time=6095.899..6095.900 rows=1 loops=1)
           ->  Subquery Scan revenue0  (cost=221672.77..221678.50 rows=327
width=8) (actual time=6054.701..6086.695 rows=10000 loops=1)
                 ->  HashAggregate  (cost=221672.77..221675.23 rows=327 width=12)
(actual time=6054.695..6071.681 rows=10000 loops=1)
                       ->  Seq Scan on lineitem  (cost=0.00..220616.23 rows=211310
width=12) (actual time=0.068..5477.123 rows=224814 loops=1)
                             Filter: ((l_shipdate >= '1997-02-01'::date) AND
(l_shipdate < '1997-05-02 00:00:00'::timestamp without time zone))
   ->  Index Scan using supplier_pkey on supplier  (cost=0.00..388.00 rows=10000
width=81) (actual time=5.927..7.537 rows=966 loops=1)
   ->  Sort  (cost=222223.70..222224.52 rows=327 width=12) (actual
time=12118.511..12118.513 rows=1 loops=1)
         Sort Key: revenue0.supplier_no
         ->  Subquery Scan revenue0  (cost=222201.05..222210.04 rows=327 width=12)
(actual time=12115.880..12118.494 rows=1 loops=1)
               ->  HashAggregate  (cost=222201.05..222206.77 rows=327 width=12)
(actual time=12115.874..12118.482 rows=1 loops=1)
                     Filter: (sum((l_extendedprice * (1::double precision -
l_discount))) = $0)
                     ->  Seq Scan on lineitem  (cost=0.00..220616.23 rows=211310
width=12) (actual time=0.081..5426.073 rows=224814 loops=1)
                           Filter: ((l_shipdate >= '1997-02-01'::date) AND
(l_shipdate < '1997-05-02 00:00:00'::timestamp without time zone))
 Total runtime: 12128.817 ms
(17 rows)
```

なお、こういったケースでは、view を materialized してインデックスを付与することによって大きな改善が見込める。

```
create table m_revenule0 as select * from revenue0;
create index i_total_revenue on m_revenue0(total_revenue);
explain analyze select s_suppkey, s_name, s_address, s_phone, total_revenue from
supplier, m_revenue0 where s_suppkey = supplier_no and total_revenue = (select
total_revenue from m_revenue0 order by total_revenue desc limit 1) order by s_suppkey;
```

リスト 2.15-4 に問い合わせと実行計画を示す。

リスト 2.15-4　PostgreSQL8.0 での materialized 化後の実行計画

```
select s_suppkey, s_name, s_address, s_phone, total_revenue from supplier,
m_revenue0 where s_suppkey = supplier_no and total_revenue = (select total_revenue
from m_revenue0 order by total_revenue desc limit 1) order by s_suppkey;
```

```
Sort  (cost=260.77..260.90 rows=50 width=89) (actual time=0.266..0.267 rows=1
loops=1)
   Sort Key: supplier.s_suppkey
   InitPlan
     ->  Limit  (cost=0.00..0.03 rows=1 width=8) (actual time=0.138..0.139 rows=1
loops=1)
           ->  Index Scan Backward using i_total_revenue on m_revenue0
(cost=0.00..317.00 rows=10000 width=8) (actual time=0.135..0.135 rows=1 loops=1)
   ->  Nested Loop  (cost=0.00..259.33 rows=50 width=89) (actual time=0.245..0.253
rows=1 loops=1)
         ->  Index Scan using i_total_revenue on m_revenue0  (cost=0.00..108.03
rows=50 width=12) (actual time=0.158..0.160 rows=1 loops=1)
               Index Cond: (total_revenue = $0)
         ->  Index Scan using supplier_pkey on supplier  (cost=0.00..3.01 rows=1
width=81) (actual time=0.079..0.082 rows=1 loops=1)
               Index Cond: (supplier.s_suppkey = "outer".supplier_no)
 Total runtime: 0.359 ms
(11 rows)
```

2.16　Q16

問い合わせをリスト 2.16-1 に示す。

リスト 2.16-1　Q16 の問い合わせ

```
select p_brand, p_type, p_size, count(distinct ps_suppkey) as supplier_cnt from
partsupp, part where p_partkey = ps_partkey and p_brand <> 'Brand#22' and p_type not
like 'MEDIUM ANODIZED%' and p_size in (34, 48, 39, 50, 14, 19, 9, 16) and ps_suppkey
not in ( select s_suppkey from supplier where s_comment like '%Customer%Complaints%' )
group by p_brand, p_type, p_size order by supplier_cnt desc, p_brand, p_type, p_size;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.16-2 に示す。

リスト 2.16-2　Q16 の実行計画

```
Sort  (cost=53864.30..53933.27 rows=27589 width=46) (actual
time=6553.061..6570.626 rows=18376 loops=1)
   Sort Key: count(DISTINCT partsupp.ps_suppkey), part.p_brand, part.p_type,
part.p_size
   ->  GroupAggregate  (cost=50522.57..51329.01 rows=27589 width=46) (actual
time=5903.064..6338.733 rows=18376 loops=1)
         ->  Sort  (cost=50522.57..50670.07 rows=58997 width=46) (actual
time=5902.939..6022.236 rows=119279 loops=1)
               Sort Key: part.p_brand, part.p_type, part.p_size
               ->  Merge Join  (cost=366.48..44669.75 rows=58997 width=46) (actual
time=41.384..4201.771 rows=119279 loops=1)
                     Merge Cond: ("outer".p_partkey = "inner".ps_partkey)
                     ->  Index Scan using part_pkey on part  (cost=0.00..12571.17
rows=29437 width=46) (actual time=0.185..904.224 rows=29832 loops=1)
                           Filter: ((p_brand <> 'Brand#22'::bpchar) AND
((p_type)::text !~~ 'MEDIUM ANODIZED%'::text) AND ((p_size = 34) OR (p_size = 48)
OR (p_size = 39) OR (p_size = 50) OR (p_size = 14) OR (p_size = 19) OR (p_size = 9)
OR (p_size = 16)))
                     ->  Index Scan using i_ps_partkey on partsupp
(cost=366.48..30459.45 rows=400959 width=8) (actual time=41.119..2215.723
rows=799661 loops=1)
                           Filter: (NOT (hashed subplan))
                           SubPlan
```

```
                          ->  Seq Scan on supplier  (cost=0.00..366.48 rows=1
width=4) (actual time=1.468..40.972 rows=4 loops=1)
                                Filter: ((s_comment)::text ~~
'%Customer%Complaints%'::text)
 Total runtime: 6595.502 ms
(15 rows)
```

sort_mem を 102400 に増やして hash join を採用させたところ、改善が見られた(リスト 2.16-3)。

リスト 2.16-3　sort_mem 設定後の実行計画

```
Sort  (cost=50532.46..50599.31 rows=26740 width=46) (actual
time=5313.257..5324.501 rows=18376 loops=1)
   Sort Key: count(DISTINCT partsupp.ps_suppkey), part.p_brand, part.p_type,
part.p_size
   ->  GroupAggregate  (cost=47786.26..48566.17 rows=26740 width=46) (actual
time=4823.398..5225.736 rows=18376 loops=1)
         ->  Sort  (cost=47786.26..47928.87 rows=57045 width=46) (actual
time=4823.302..4900.203 rows=119279 loops=1)
               Sort Key: part.p_brand, part.p_type, part.p_size
               ->  Hash Join  (cost=12194.31..43279.76 rows=57045 width=46)
(actual time=782.295..4089.971 rows=119279 loops=1)
                     Hash Cond: ("outer".ps_partkey = "inner".p_partkey)
                     ->  Seq Scan on partsupp  (cost=366.00..27881.00 rows=400000
width=8) (actual time=41.228..2371.567 rows=799680 loops=1)
                           Filter: (NOT (hashed subplan))
                           SubPlan
                             ->  Seq Scan on supplier  (cost=0.00..366.00 rows=1
width=4) (actual time=1.482..41.123 rows=4 loops=1)
                                   Filter: ((s_comment)::text ~~
'%Customer%Complaints%'::text)
                     ->  Hash  (cost=11757.00..11757.00 rows=28522 width=46)
(actual time=740.454..740.454 rows=0 loops=1)
                           ->  Seq Scan on part  (cost=0.00..11757.00 rows=28522
width=46) (actual time=0.100..700.791 rows=29832 loops=1)
                                 Filter: ((p_brand <> 'Brand#22'::bpchar) AND
((p_type)::text !~~ 'MEDIUM ANODIZED%'::text) AND ((p_size = 34) OR (p_size = 48)
OR (p_size = 39) OR (p_size = 50) OR (p_size = 14) OR (p_size = 19) OR (p_size = 9)
```

```
OR (p_size = 16)))
 Total runtime: 5371.763 ms
(16 rows)
```

PostgreSQL 8.0 での実行計画をリスト 2.16-4 に示す。

リスト 2.16-4　PostgreSQL8.0 での実行計画

```
Sort  (cost=54171.31..54237.98 rows=26668 width=46) (actual
time=8744.892..8763.445 rows=18376 loops=1)
   Sort Key: count(DISTINCT partsupp.ps_suppkey), part.p_brand, part.p_type,
part.p_size
   ->  GroupAggregate  (cost=50914.25..51691.95 rows=26668 width=46) (actual
time=7763.568..8355.674 rows=18376 loops=1)
         ->  Sort  (cost=50914.25..51056.45 rows=56883 width=46) (actual
time=7763.410..7888.540 rows=119279 loops=1)
               Sort Key: part.p_brand, part.p_type, part.p_size
               ->  Merge Join  (cost=371.00..45144.91 rows=56883 width=46) (actual
time=42.318..3709.562 rows=119279 loops=1)
                     Merge Cond: ("outer".p_partkey = "inner".ps_partkey)
                     ->  Index Scan using part_pkey on part  (cost=0.00..12675.00
rows=28445 width=46) (actual time=0.145..787.800 rows=29832 loops=1)
                           Filter: ((p_brand <> 'Brand#22'::bpchar) AND
((p_type)::text !~~ 'MEDIUM ANODIZED%'::text) AND ((p_size = 34) OR (p_size = 48)
OR (p_size = 39) OR (p_size = 50) OR (p_size = 14) OR (p_size = 19) OR (p_size = 9)
OR (p_size = 16)))
                     ->  Index Scan using i_ps_partkey on partsupp
(cost=371.00..30839.52 rows=399945 width=8) (actual time=42.099..1886.817
rows=799661 loops=1)
                           Filter: (NOT (hashed subplan))
                           SubPlan
                             ->  Seq Scan on supplier  (cost=0.00..371.00 rows=1
width=4) (actual time=1.425..41.946 rows=4 loops=1)
                                   Filter: ((s_comment)::text ~~
'%Customer%Complaints%'::text)
 Total runtime: 8787.034 ms
(15 rows)
```

7.4 同様、sort_mem を増やして改善された。

リスト 2.16-5　PostgreSQL8.0 での改善後の実行計画

```
Sort  (cost=50019.01..50085.68 rows=26668 width=46) (actual
time=6144.285..6155.574 rows=18376 loops=1)
   Sort Key: count(DISTINCT partsupp.ps_suppkey), part.p_brand, part.p_type,
part.p_size
   ->  GroupAggregate  (cost=47280.83..48058.54 rows=26668 width=46) (actual
time=5423.501..5966.945 rows=18376 loops=1)
         ->  Sort  (cost=47280.83..47423.04 rows=56883 width=46) (actual
time=5423.353..5500.853 rows=119279 loops=1)
               Sort Key: part.p_brand, part.p_type, part.p_size
               ->  Hash Join  (cost=12305.11..42788.29 rows=56883 width=46)
(actual time=774.579..3221.745 rows=119279 loops=1)
                     Hash Cond: ("outer".ps_partkey = "inner".p_partkey)
                     ->  Seq Scan on partsupp  (cost=371.00..28285.63 rows=399945
width=8) (actual time=42.346..1544.700 rows=799680 loops=1)
                           Filter: (NOT (hashed subplan))
                           SubPlan
                             ->  Seq Scan on supplier  (cost=0.00..371.00 rows=1
width=4) (actual time=1.479..42.239 rows=4 loops=1)
                                   Filter: ((s_comment)::text ~~
'%Customer%Complaints%'::text)
                     ->  Hash  (cost=11863.00..11863.00 rows=28445 width=46)
(actual time=731.634..731.634 rows=0 loops=1)
                           ->  Seq Scan on part  (cost=0.00..11863.00 rows=28445
width=46) (actual time=0.102..689.858 rows=29832 loops=1)
                                 Filter: ((p_brand <> 'Brand#22'::bpchar) AND
((p_type)::text !~~ 'MEDIUM ANODIZED%'::text) AND ((p_size = 34) OR (p_size = 48)
OR (p_size = 39) OR (p_size = 50) OR (p_size = 14) OR (p_size = 19) OR (p_size = 9)
OR (p_size = 16)))
 Total runtime: 6202.329 ms
(16 rows)
```

2.17　Q17

問い合わせをリスト 2.17-1 に示す。

リスト 2.17-1　Q17 の問い合わせ

```
select sum(l_extendedprice) / 7.0 as avg_yearly from lineitem, part where p_partkey
= l_partkey and p_brand = 'Brand#22' and p_container = 'JUMBO BAG' and l_quantity
< ( select 0.2 * avg(l_quantity) from lineitem where l_partkey = p_partkey );
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.17-2 に示す。

リスト 2.17-2　Q17 の実行計画

```
Aggregate  (cost=819670.45..819670.45 rows=1 width=4) (actual
time=1405.908..1405.909 rows=1 loops=1)
   ->  Nested Loop  (cost=0.00..819665.50 rows=1979 width=4) (actual
time=7.392..1404.948 rows=541 loops=1)
         Join Filter: ("inner".l_quantity < (subplan))
         ->  Seq Scan on part  (cost=0.00..7757.93 rows=198 width=4) (actual
time=0.641..216.879 rows=200 loops=1)
               Filter: ((p_brand = 'Brand#22'::bpchar) AND (p_container = 'JUMBO
BAG'::bpchar))
         ->  Index Scan using i_l_partkey on lineitem  (cost=0.00..127.23 rows=31
width=12) (actual time=1.709..1.778 rows=31 loops=200)
               Index Cond: ("outer".p_partkey = lineitem.l_partkey)
         SubPlan
           ->  Aggregate  (cost=127.31..127.31 rows=1 width=4) (actual
time=0.131..0.132 rows=1 loops=6120)
                 ->  Index Scan using i_l_partkey on lineitem  (cost=0.00..127.23
rows=32 width=4) (actual time=0.007..0.086 rows=32 loops=6120)
                       Index Cond: (l_partkey = $0)
 Total runtime: 1406.273 ms
(12 rows)
```

この問い合せでは、part テーブルに順スキャンが行われている(-> Seq Scanon part (cost=0.00..7757.93 rows=198 width=4) (actual time=0.641..216.879 rows=200 loops=1))。

これをインデックスを使うように改善することを考える。対応する検索条件が(p_brand = 'Brand#22' and p_container = 'JUMBO BAG')であるので、p_brand 列と p_container 列にインデックスを張ることも考えられるが、それぞれの条件にマッチする行数が多すぎるため、インデックスが使用されない。そこで、p_brand と p_container のマルチカラムインデックスを作成する。

```
create index i_p_brand_container on part(p_brand, p_container);
```

これにより改善された結果をリスト 2.17-3 に示す。

リスト 2.17-3　改善された問い合わせ計画

```
Aggregate  (cost=621839.82..621839.82 rows=1 width=4) (actual
time=867.362..867.363 rows=1 loops=1)
   ->  Nested Loop  (cost=0.00..621835.32 rows=1796 width=4) (actual
time=3.967..866.390 rows=535 loops=1)
         Join Filter: ("inner".l_quantity < (subplan))
         ->  Index Scan using i_p_brand_container on part  (cost=0.00..781.24
rows=198 width=4) (actual time=0.161..6.697 rows=200 loops=1)
               Index Cond: ((p_brand = 'Brand#22'::bpchar) AND (p_container = 'JUMBO
BAG'::bpchar))
         ->  Index Scan using i_l_partkey on lineitem  (cost=0.00..111.14 rows=27
width=12) (actual time=0.053..0.123 rows=31 loops=200)
               Index Cond: ("outer".p_partkey = lineitem.l_partkey)
         SubPlan
           ->  Aggregate  (cost=111.21..111.22 rows=1 width=4) (actual
time=0.132..0.132 rows=1 loops=6114)
                 ->  Index Scan using i_l_partkey on lineitem  (cost=0.00..111.14
rows=28 width=4) (actual time=0.007..0.086 rows=31 loops=6114)
                       Index Cond: (l_partkey = $0)
 Total runtime: 867.498 ms
(12 rows)
```

PostgreSQL 8.0 でも同様の改善が可能である．改善前の問い合わせ計画をリスト 2.17-4 に示す。

リスト 2.17-4　PostgreSQL8.0 での問い合わせ計画

```
Aggregate  (cost=864940.93..864940.93 rows=1 width=4) (actual
time=1534.475..1534.476 rows=1 loops=1)
   ->  Nested Loop  (cost=0.00..864935.95 rows=1991 width=4) (actual
time=8.048..1533.510 rows=533 loops=1)
         Join Filter: ("inner".l_quantity < (subplan))
         ->  Seq Scan on part  (cost=0.00..7863.00 rows=199 width=4) (actual
```

```
time=0.880..293.053 rows=200 loops=1)
              Filter: ((p_brand = 'Brand#22'::bpchar) AND (p_container = 'JUMBO
BAG'::bpchar))
        ->  Index Scan using i_l_partkey on lineitem  (cost=0.00..130.42 rows=32
width=12) (actual time=2.025..2.093 rows=31 loops=200)
              Index Cond: ("outer".p_partkey = lineitem.l_partkey)
        SubPlan
          ->  Aggregate  (cost=130.50..130.50 rows=1 width=4) (actual
time=0.130..0.130 rows=1 loops=6107)
                ->  Index Scan using i_l_partkey on lineitem  (cost=0.00..130.42
rows=32 width=4) (actual time=0.008..0.086 rows=31 loops=6107)
                      Index Cond: (l_partkey = $0)
 Total runtime: 1534.884 ms
(12 rows)
```

次のインデックスを設定した後の問い合わせ計画をリスト 2.17-5 に示す。

```
create index i_p_brand_container on part(p_brand, p_container);
```

リスト 2.17-5　PostgreSQL8.0 での改善後の問い合わせ計画

```
Aggregate  (cost=857862.49..857862.50 rows=1 width=4) (actual
time=843.343..843.344 rows=1 loops=1)
   ->  Nested Loop  (cost=0.00..857857.51 rows=1991 width=4) (actual
time=3.377..842.359 rows=533 loops=1)
        Join Filter: ("inner".l_quantity < (subplan))
        ->  Index Scan using i_p_brand_container on part  (cost=0.00..784.57
rows=199 width=4) (actual time=0.197..6.425 rows=200 loops=1)
              Index Cond: ((p_brand = 'Brand#22'::bpchar) AND (p_container = 'JUMBO
BAG'::bpchar))
        ->  Index Scan using i_l_partkey on lineitem  (cost=0.00..130.42 rows=32
width=12) (actual time=0.045..0.113 rows=31 loops=200)
              Index Cond: ("outer".p_partkey = lineitem.l_partkey)
        SubPlan
          ->  Aggregate  (cost=130.50..130.50 rows=1 width=4) (actual
time=0.129..0.129 rows=1 loops=6107)
                ->  Index Scan using i_l_partkey on lineitem  (cost=0.00..130.42
rows=32 width=4) (actual time=0.008..0.085 rows=31 loops=6107)
                      Index Cond: (l_partkey = $0)
```

```
 Total runtime: 843.493 ms
(12 rows)
```

2.18　Q18

問い合わせをリスト 2.18-1 に示す。

リスト 2.18-1　Q18 の問い合わせ

```
select c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice, sum(l_quantity)
from customer, orders, lineitem where o_orderkey in ( select l_orderkey from lineitem
group by l_orderkey having sum(l_quantity) > 313 ) and c_custkey = o_custkey and
o_orderkey = l_orderkey group by c_name, c_custkey, o_orderkey, o_orderdate,
o_totalprice order by o_totalprice desc, o_orderdate LIMIT 100;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.18-2 に示す。

リスト 2.18-2　Q18 の実行計画

```
Limit  (cost=267201.27..267201.52 rows=100 width=42) (actual
time=19502.400..19502.417 rows=9 loops=1)
   ->  Sort  (cost=267201.27..267203.11 rows=737 width=42) (actual
time=19502.397..19502.403 rows=9 loops=1)
         Sort Key: orders.o_totalprice, orders.o_orderdate
         ->  HashAggregate  (cost=267164.32..267166.17 rows=737 width=42) (actual
time=19502.311..19502.330 rows=9 loops=1)
               ->  Nested Loop  (cost=265320.71..267153.27 rows=737 width=42)
(actual time=19500.381..19502.115 rows=63 loops=1)
                     ->  Nested Loop  (cost=265320.71..266531.11 rows=201
width=42) (actual time=19500.273..19501.207 rows=9 loops=1)
                           ->  Nested Loop  (cost=265320.71..265925.91 rows=200
width=20) (actual time=19500.117..19500.670 rows=9 loops=1)
                                 ->  HashAggregate  (cost=265320.71..265320.71
rows=200 width=4) (actual time=19499.968..19500.006 rows=9 loops=1)
                                       ->  Subquery Scan "IN_subquery"
(cost=0.00..261231.73 rows=1635593 width=4) (actual time=4237.225..19499.918 rows=9
loops=1)
                                             ->  GroupAggregate
```

```
(cost=0.00..244875.80 rows=1635593 width=8) (actual time=4237.220..19499.874 rows=9
loops=1)
                                                  Filter: (sum(l_quantity) >
313::double precision)
                                                  ->  Index Scan using
i_l_orderkey on lineitem  (cost=0.00..206716.32 rows=5996302 width=8) (actual
time=3.464..11906.768 rows=6000935 loops=1)
                                 ->  Index Scan using orders_pkey on orders
(cost=0.00..3.01 rows=1 width=16) (actual time=0.064..0.066 rows=1 loops=9)
                                       Index Cond: (orders.o_orderkey =
"outer".l_orderkey)
                           ->  Index Scan using customer_pkey on customer
(cost=0.00..3.01 rows=1 width=26) (actual time=0.052..0.054 rows=1 loops=9)
                                 Index Cond: (customer.c_custkey =
"outer".o_custkey)
                     ->  Index Scan using i_l_orderkey on lineitem
(cost=0.00..3.05 rows=4 width=8) (actual time=0.060..0.080 rows=7 loops=9)
                           Index Cond: (lineitem.l_orderkey = "outer".l_orderkey)
 Total runtime: 19502.971 ms
(19 rows)
```

特に改善する余地はなかった。

PostgreSQL 8.0 の実行計画をリスト 2.18-3 に示す。7.4 との大きな違いはない．こちらも特に改善の余地はなかった。

リスト 2.18-3　PostgreSQL8.0 の実行計画

```
Limit  (cost=569883.61..569883.86 rows=100 width=42) (actual
time=30781.664..30781.683 rows=9 loops=1)
   ->  Sort  (cost=569883.61..570221.11 rows=134999 width=42) (actual
time=30781.662..30781.669 rows=9 loops=1)
         Sort Key: orders.o_totalprice, orders.o_orderdate
         ->  GroupAggregate  (cost=551244.97..553944.95 rows=134999 width=42)
(actual time=30781.458..30781.635 rows=9 loops=1)
               ->  Sort  (cost=551244.97..551582.47 rows=134999 width=42) (actual
time=30781.420..30781.465 rows=63 loops=1)
                     Sort Key: customer.c_name, customer.c_custkey,
orders.o_orderkey, orders.o_orderdate, orders.o_totalprice
                     ->  Merge Join  (cost=308525.58..535306.31 rows=134999
```

```
width=42) (actual time=22655.181..30781.063 rows=63 loops=1)
                          Merge Cond: ("outer".l_orderkey = "inner".o_orderkey)
                          ->  Index Scan using i_l_orderkey on lineitem
(cost=0.00..209753.36 rows=6001215 width=8) (actual time=0.125..6867.615
rows=4807580 loops=1)
                          ->  Sort  (cost=308525.58..308609.93 rows=33739
width=42) (actual time=19762.625..19762.670 rows=57 loops=1)
                                Sort Key: orders.o_orderkey
                                ->  Merge Join  (cost=298336.52..305354.75
rows=33739 width=42) (actual time=19453.828..19762.594 rows=9 loops=1)
                                      Merge Cond: ("outer".c_custkey =
"inner".o_custkey)
                                      ->  Index Scan using customer_pkey on
customer  (cost=0.00..6138.01 rows=150001 width=26) (actual time=0.085..225.386
rows=147198 loops=1)
                                      ->  Sort  (cost=298336.52..298420.87
rows=33739 width=20) (actual time=19416.901..19416.911 rows=9 loops=1)
                                            Sort Key: orders.o_custkey
                                            ->  Merge IN Join
(cost=240397.53..295391.34 rows=33739 width=20) (actual time=4429.021..19416.863
rows=9 loops=1)
                                                  Merge Cond:
("outer".o_orderkey = "inner".l_orderkey)
                                                  ->  Index Scan using
orders_pkey on orders  (cost=0.00..50353.12 rows=1499829 width=16) (actual
time=0.100..1778.329 rows=1201687 loops=1)
                                                  ->  Materialize
(cost=240397.53..240866.92 rows=33739 width=4) (actual time=3667.088..16668.018
rows=9 loops=1)
                                                        ->  Subquery Scan
"IN_subquery"  (cost=0.00..240265.53 rows=33739 width=4) (actual
time=3667.083..16667.971 rows=9 loops=1)
                                                              ->
GroupAggregate  (cost=0.00..239928.14 rows=33739 width=8) (actual
time=3667.076..16667.918 rows=9 loops=1)
                                                                    Filter:
(sum(l_quantity) > 313::double precision)
                                                                    ->  Index
Scan using i_l_orderkey on lineitem  (cost=0.00..209753.36 rows=6001215 width=8)
(actual time=0.011..9295.362 rows=6001215 loops=1)
```

```
 Total runtime: 30781.897 ms
(25 rows)
```

2.19 Q19

問い合わせをリスト 2.19-1 に示す。

リスト 2.19-1 Q19 の問い合わせ

```
select sum(l_extendedprice* (1 - l_discount)) as revenue from lineitem, part where
p_partkey = l_partkey and l_shipmode in ('AIR', 'AIR REG') and l_shipinstruct =
'DELIVER IN PERSON' and ( ( p_brand = 'Brand#45' and p_container in ('SM CASE', 'SM
BOX', 'SM PACK', 'SM PKG') and l_quantity >= 8 and l_quantity <= 8+10 and p_size between
1 and 5 ) or ( p_brand = 'Brand#45' and p_container in ('MED BAG', 'MED BOX', 'MED
PKG', 'MED PACK') and l_quantity >= 16 and l_quantity <= 16+10 and p_size between
1 and 10 ) or ( p_brand = 'Brand#22' and p_container in ('LG CASE', 'LG BOX', 'LG
PACK', 'LG PKG') and l_quantity >= 28 and l_quantity <= 28+10 and p_size between 1
and 15 ) );
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.19-2 に示す。

リスト 2.19-2 Q19 の実行計画

```
Aggregate  (cost=278537.67..278537.67 rows=1 width=8) (actual
time=13149.178..13149.179 rows=1 loops=1)
   ->  Merge Join  (cost=253595.35..278537.42 rows=97 width=8) (actual
time=11689.635..13148.516 rows=105 loops=1)
         Merge Cond: ("outer".p_partkey = "inner".l_partkey)
         Join Filter: ((("outer".p_brand = 'Brand#45'::bpchar) AND
(("outer".p_container = 'SM CASE'::bpchar) OR ("outer".p_container = 'SM
BOX'::bpchar) OR ("outer".p_container = 'SM PACK'::bpchar) OR ("outer".p_container
= 'SM PKG'::bpchar)) AND ("inner".l_quantity >= 8::double precision) AND
("inner".l_quantity <= 18::double precision) AND ("outer".p_size >= 1) AND
("outer".p_size <= 5)) OR (("outer".p_brand = 'Brand#45'::bpchar) AND
(("outer".p_container = 'MED BAG'::bpchar) OR ("outer".p_container = 'MED
BOX'::bpchar) OR ("outer".p_container = 'MED PKG'::bpchar) OR ("outer".p_container
= 'MED PACK'::bpchar)) AND ("inner".l_quantity >= 16::double precision) AND
("inner".l_quantity <= 26::double precision) AND ("outer".p_size >= 1) AND
```

```
("outer".p_size <= 10)) OR (("outer".p_brand = 'Brand#22'::bpchar) AND
(("outer".p_container = 'LG CASE'::bpchar) OR ("outer".p_container = 'LG
BOX'::bpchar) OR ("outer".p_container = 'LG PACK'::bpchar) OR ("outer".p_container
= 'LG PKG'::bpc!
har)) AND ("inner".l_quantity >= 28::double precision) AND ("inner".l_quantity <=
38::double precision) AND ("outer".p_size >= 1) AND ("outer".p_size <= 15)))
        ->  Index Scan using part_pkey on part  (cost=0.00..7569.62 rows=200062
width=36) (actual time=0.124..416.100 rows=200000 loops=1)
        ->  Sort  (cost=253595.35..254106.67 rows=204525 width=16) (actual
time=11685.680..11901.554 rows=214421 loops=1)
              Sort Key: lineitem.l_partkey
              ->  Seq Scan on lineitem  (cost=0.00..232559.29 rows=204525
width=16) (actual time=11.418..9839.689 rows=214421 loops=1)
                    Filter: (((l_shipmode = 'AIR'::bpchar) OR (l_shipmode = 'AIR
REG'::bpchar)) AND (l_shipinstruct = 'DELIVER IN PERSON'::bpchar))
 Total runtime: 13160.495 ms
(10 rows)
```

この問い合わせ計画では、

```
l_shipmode in ('AIR', 'AIR REG') and l_shipinstruct = 'DELIVER IN PERSON'
```

に対応する以下の問い合わせ計画:

```
                    Filter: (((l_shipmode = 'AIR'::bpchar) OR
                    (l_shipmode = 'AIR REG'::bpchar)) AND
                    (l_shipinstruct = 'DELIVER IN PERSON'::bpchar))
```

でインデックスが使われていない．そこで次のようにインデックスを設定し、

```
create index i_l_shipinstruct on lineitem(l_shipinstruct);
```

set enable_seqscan to off;にしてインデックスを使用させたところ、改善が見られた(リスト 2.19-3)。

リスト 2.19-3　改善後の問い合わせ計画

```
Aggregate  (cost=3468594.25..3468594.26 rows=1 width=8) (actual
time=9203.082..9203.082 rows=1 loops=1)
```

```
   ->  Merge Join  (cost=3444468.77..3468594.02 rows=94 width=8) (actual
time=7280.185..9202.548 rows=105 loops=1)
         Merge Cond: ("outer".l_partkey = "inner".p_partkey)
         Join Filter: ((("inner".p_brand = 'Brand#45'::bpchar) AND
(("inner".p_container = 'SM CASE'::bpchar) OR ("inner".p_container = 'SM
BOX'::bpchar) OR ("inner".p_container = 'SM PACK'::bpchar) OR ("inner".p_container
= 'SM PKG'::bpchar)) AND ("outer".l_quantity >= 8::double precision) AND
("outer".l_quantity <= 18::double precision) AND ("inner".p_size >= 1) AND
("inner".p_size <= 5)) OR (("inner".p_brand = 'Brand#45'::bpchar) AND
(("inner".p_container = 'MED BAG'::bpchar) OR ("inner".p_container = 'MED
BOX'::bpchar) OR ("inner".p_container = 'MED PKG'::bpchar) OR ("inner".p_container
= 'MED PACK'::bpchar)) AND ("outer".l_quantity >= 16::double precision) AND
("outer".l_quantity <= 26::double precision) AND ("inner".p_size >= 1) AND
("inner".p_size <= 10)) OR (("inner".p_brand = 'Brand#22'::bpchar) AND
(("inner".p_container = 'LG CASE'::bpchar) OR ("inner".p_container = 'LG
BOX'::bpchar) OR ("inner".p_container = 'LG PACK'::bpchar) OR ("inner".p_container
= 'LG PKG'::bpc!
har)) AND ("outer".l_quantity >= 28::double precision) AND ("outer".l_quantity <=
38::double precision) AND ("inner".p_size >= 1) AND ("inner".p_size <= 15)))
         ->  Sort  (cost=3444468.77..3445008.38 rows=215845 width=16) (actual
time=7275.091..7503.188 rows=214592 loops=1)
               Sort Key: lineitem.l_partkey
               ->  Index Scan using i_l_shipmode, i_l_shipmode on lineitem
(cost=0.00..3422075.30 rows=215845 width=16) (actual time=10.080..5380.177
rows=214592 loops=1)
                     Index Cond: ((l_shipmode = 'AIR'::bpchar) OR (l_shipmode = 'AIR
REG'::bpchar))
                     Filter: (l_shipinstruct = 'DELIVER IN PERSON'::bpchar)
         ->  Index Scan using part_pkey on part  (cost=0.00..7398.00 rows=200000
width=36) (actual time=0.134..583.304 rows=282958 loops=1)
 Total runtime: 9213.211 ms
(11 rows)
```

この問い合わせは、PostgreSQL 7.4 では実行時間がかかりすぎ、事実上実行が不可能なために DBT-3 本来の問い合わせから書き換えたものになっている。

PostgreSQL 8.0 では、リスト 2.19-4 のような本来の問い合わせを実行できる。

リスト 2.19-4　本来の問い合わせ

```
select
        sum(l_extendedprice* (1 - l_discount)) as revenue
from
        lineitem,
        part
where
        (
                p_partkey = l_partkey
                and p_brand = 'Brand#14'
                and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
                and l_quantity >= 6 and l_quantity <= 6+10
                and p_size between 1 and 5
                and l_shipmode in ('AIR', 'AIR REG')
                and l_shipinstruct = 'DELIVER IN PERSON'
        )
        or
        (
                p_partkey = l_partkey
                and p_brand = 'Brand#23'
                and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
                and l_quantity >= 11 and l_quantity <= 11+10
                and p_size between 1 and 10
                and l_shipmode in ('AIR', 'AIR REG')
                and l_shipinstruct = 'DELIVER IN PERSON'
        )
        or
        (
                p_partkey = l_partkey
                and p_brand = 'Brand#14'
                and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
                and l_quantity >= 29 and l_quantity <= 29+10
                and p_size between 1 and 15
                and l_shipmode in ('AIR', 'AIR REG')
                and l_shipinstruct = 'DELIVER IN PERSON'
        );
```

この問い合わせに対する問い合わせ計画をリスト 2.19-5 に示す。

リスト 2.19-5　PostgreSQL8.0 による問い合わせ計画

```
Aggregate  (cost=252089.66..252089.67 rows=1 width=8) (actual
time=11814.093..11814.094 rows=1 loops=1)
   ->  Hash Join  (cost=9101.63..252089.41 rows=99 width=8) (actual
time=1042.025..11813.303 rows=110 loops=1)
         Hash Cond: ("outer".l_partkey = "inner".p_partkey)
         Join Filter: ((("inner".p_brand = 'Brand#14'::bpchar) AND
(("inner".p_container = 'SM CASE'::bpchar) OR ("inner".p_container = 'SM
BOX'::bpchar) OR ("inner".p_container = 'SM PACK'::bpchar) OR ("inner".p_container
= 'SM PKG'::bpchar)) AND ("outer".l_quantity >= 6::double precision) AND
("outer".l_quantity <= 16::double precision) AND ("inner".p_size <= 5)) OR
(("inner".p_brand = 'Brand#23'::bpchar) AND (("inner".p_container = 'MED
BAG'::bpchar) OR ("inner".p_container = 'MED BOX'::bpchar) OR ("inner".p_container
= 'MED PKG'::bpchar) OR ("inner".p_container = 'MED PACK'::bpchar)) AND
("outer".l_quantity >= 11::double precision) AND ("outer".l_quantity <= 21::double
precision) AND ("inner".p_size <= 10)) OR (("inner".p_brand = 'Brand#14'::bpchar)
AND (("inner".p_container = 'LG CASE'::bpchar) OR ("inner".p_container = 'LG
BOX'::bpchar) OR ("inner".p_container = 'LG PACK'::bpchar) OR ("inner".p_container
= 'LG PKG'::bpchar)) AND ("outer".l_quantity >= 29::double precisio!
n) AND ("outer".l_quantity <= 39::double precision) AND ("inner".p_size <= 15)))
         ->  Seq Scan on lineitem  (cost=0.00..235619.26 rows=212568 width=16)
(actual time=0.083..9803.914 rows=214377 loops=1)
               Filter: (((l_shipmode = 'AIR'::bpchar) OR (l_shipmode = 'AIR
REG'::bpchar)) AND (l_shipinstruct = 'DELIVER IN PERSON'::bpchar))
         ->  Hash  (cost=8863.00..8863.00 rows=23051 width=36) (actual
time=611.401..611.401 rows=0 loops=1)
               ->  Seq Scan on part  (cost=0.00..8863.00 rows=23051 width=36)
(actual time=0.102..582.868 rows=15923 loops=1)
                     Filter: ((p_size >= 1) AND ((p_brand = 'Brand#14'::bpchar) OR
(p_brand = 'Brand#23'::bpchar) OR (p_brand = 'Brand#14'::bpchar)))
 Total runtime: 11814.724 ms
(10 rows)
```

7.4 と同様に、

```
create index i_l_shipinstruct on lineitem(l_shipinstruct);
```

としてインデックスを設定してみたが、特に改善はなかった。

2.20　Q20

問い合わせをリスト 2.20-1 に示す。

リスト 2.20-1　Q20 の問い合わせ

```
select s_name, s_address from supplier, nation where s_suppkey in ( select distinct
(ps_suppkey) from partsupp, part where ps_partkey=p_partkey and p_name like 'violet%'
and ps_availqty > ( select 0.5 * sum(l_quantity) from lineitem where l_partkey =
ps_partkey and l_suppkey = ps_suppkey and l_shipdate >= '1996-01-01' and l_shipdate
< date '1996-01-01' + interval '1 year' ) ) and s_nationkey = n_nationkey and n_name
= 'UNITED STATES' order by s_name;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.20-2 に示す。

リスト 2.20-2　Q20 の実行計画

```
Sort  (cost=7293.78..7293.78 rows=1 width=57) (actual time=1478.929..1479.013
rows=137 loops=1)
   Sort Key: supplier.s_name
   ->  Hash Join  (cost=7287.69..7293.77 rows=1 width=57) (actual
time=1427.640..1478.601 rows=137 loops=1)
         Hash Cond: ("outer".s_nationkey = "inner".n_nationkey)
         ->  Nested Loop  (cost=7286.37..7292.42 rows=3 width=61) (actual
time=1426.015..1474.720 rows=3522 loops=1)
               ->  HashAggregate  (cost=7286.37..7286.37 rows=2 width=4) (actual
time=1425.875..1430.147 rows=3522 loops=1)
                     ->  Subquery Scan "IN_subquery"  (cost=7286.34..7286.37
rows=2 width=4) (actual time=1408.015..1421.803 rows=3522 loops=1)
                           ->  Unique  (cost=7286.34..7286.35 rows=2 width=4)
(actual time=1408.010..1416.580 rows=3522 loops=1)
                                 ->  Sort  (cost=7286.34..7286.34 rows=2 width=4)
(actual time=1408.006..1410.779 rows=4335 loops=1)
                                       Sort Key: partsupp.ps_suppkey
                                       ->  Nested Loop  (cost=0.00..7286.33
rows=2 width=4) (actual time=0.392..1399.364 rows=4335 loops=1)
                                             ->  Seq Scan on part
(cost=0.00..7257.77 rows=1 width=4) (actual time=0.084..210.348 rows=1627 loops=1)
                                                   Filter: ((p_name)::text ~~
```

```
'violet%'::text)
                                          ->  Index Scan using i_ps_partkey on
partsupp  (cost=0.00..28.53 rows=1 width=8) (actual time=0.307..0.720 rows=3
loops=1627)
                                                Index Cond:
(partsupp.ps_partkey = "outer".p_partkey)
                                                Filter: ((ps_availqty)::double
precision > (subplan))
                                                SubPlan
                                                  ->  Aggregate
(cost=6.03..6.03 rows=1 width=4) (actual time=0.162..0.162 rows=1 loops=6508)
                                                        ->  Index Scan using
i_l_suppkey_partkey on lineitem  (cost=0.00..6.02 rows=1 width=4) (actual
time=0.098..0.155 rows=1 loops=6508)
                                                              Index Cond:
((l_partkey = $0) AND (l_suppkey = $1))
                                                              Filter:
((l_shipdate >= '1996-01-01'::date) AND ((l_shipdate)::timestamp without time zone
< '1997-01-01 00:00:00'::timestamp without time zone))
              ->  Index Scan using supplier_pkey on supplier  (cost=0.00..3.01
rows=1 width=65) (actual time=0.007..0.008 rows=1 loops=3522)
                    Index Cond: (supplier.s_suppkey = "outer".ps_suppkey)
        ->  Hash  (cost=1.31..1.31 rows=2 width=4) (actual time=0.057..0.057
rows=0 loops=1)
              ->  Seq Scan on nation  (cost=0.00..1.31 rows=2 width=4) (actual
time=0.051..0.052 rows=1 loops=1)
                    Filter: (n_name = 'UNITED STATES'::bpchar)
 Total runtime: 1480.111 ms
(27 rows)
```

この問い合わせでは、(p_name like 'violet%')に対応する問い合わせ計画:

```
    ->  Seq Scan on part  (cost=0.00..7257.77 rows=1 width=4) (actual
time=0.084..210.348 rows=1627 loops=1)
          Filter: ((p_name)::text ~~ 'violet%'::text)
```

でインデックスが使われていない。そこで、以下のようにインデックスを設定してみた。

```
create index i_p_name on part(p_name);
```

その結果、リスト 2.20-3 のように改善が見られた。

リスト 2.20-3　改善後の問い合わせ計画

```
Sort  (cost=47.47..47.48 rows=1 width=57) (actual time=1353.804..1353.889 rows=136
loops=1)
   Sort Key: supplier.s_name
   ->  Hash Join  (cost=41.38..47.46 rows=1 width=57) (actual
time=1300.952..1353.463 rows=136 loops=1)
         Hash Cond: ("outer".s_nationkey = "inner".n_nationkey)
         ->  Nested Loop  (cost=40.07..46.12 rows=3 width=61) (actual
time=1299.373..1349.527 rows=3523 loops=1)
               ->  HashAggregate  (cost=40.07..40.07 rows=2 width=4) (actual
time=1299.228..1303.670 rows=3523 loops=1)
                     ->  Subquery Scan "IN_subquery"  (cost=40.03..40.06 rows=2
width=4) (actual time=1281.055..1295.214 rows=3523 loops=1)
                           ->  Unique  (cost=40.03..40.04 rows=2 width=4) (actual
time=1281.050..1289.689 rows=3523 loops=1)
                                 ->  Sort  (cost=40.03..40.04 rows=2 width=4)
(actual time=1281.047..1283.860 rows=4336 loops=1)
                                       Sort Key: partsupp.ps_suppkey
                                       ->  Nested Loop  (cost=0.00..40.02 rows=2
width=4) (actual time=0.479..1270.803 rows=4336 loops=1)
                                             ->  Index Scan using i_p_name on part
(cost=0.00..6.01 rows=1 width=4) (actual time=0.116..47.257 rows=1627 loops=1)
                                                   Index Cond: (((p_name)::text >=
'violet'::character varying) AND ((p_name)::text < 'violeu'::character varying))
                                                   Filter: ((p_name)::text ~~
'violet%'::text)
                                             ->  Index Scan using i_ps_partkey on
partsupp  (cost=0.00..33.99 rows=2 width=8) (actual time=0.325..0.742 rows=3
loops=1627)
                                                   Index Cond:
(partsupp.ps_partkey = "outer".p_partkey)
                                                   Filter: ((ps_availqty)::double
precision > (subplan))
                                                   SubPlan
```

```
                                                    ->  Aggregate
(cost=6.03..6.03 rows=1 width=4) (actual time=0.165..0.166 rows=1 loops=6508)
                                                          ->  Index Scan using
i_l_suppkey_partkey on lineitem  (cost=0.00..6.02 rows=1 width=4) (actual
time=0.101..0.159 rows=1 loops=6508)
                                                                Index Cond:
((l_partkey = $0) AND (l_suppkey = $1))
                                                                Filter:
((l_shipdate >= '1996-01-01'::date) AND ((l_shipdate)::timestamp without time zone
< '1997-01-01 00:00:00'::timestamp without time zone))
               ->  Index Scan using supplier_pkey on supplier  (cost=0.00..3.01
rows=1 width=65) (actual time=0.007..0.009 rows=1 loops=3523)
                     Index Cond: (supplier.s_suppkey = "outer".ps_suppkey)
         ->  Hash  (cost=1.31..1.31 rows=2 width=4) (actual time=0.084..0.084
rows=0 loops=1)
               ->  Seq Scan on nation  (cost=0.00..1.31 rows=2 width=4) (actual
time=0.078..0.080 rows=1 loops=1)
                     Filter: (n_name = 'UNITED STATES'::bpchar)
 Total runtime: 1354.946 ms
(28 rows)
```

PostgreSQL 8.0 では以下のような問い合わせ計画となった(リスト 2.20-4)。

リスト 2.20-4　PostgreSQL8.0 での問い合わせ計画

```
Sort  (cost=7500.51..7500.51 rows=1 width=58) (actual time=1491.004..1491.088
rows=136 loops=1)
   Sort Key: supplier.s_name
   ->  Hash Join  (cost=7494.40..7500.50 rows=1 width=58) (actual
time=1422.585..1490.373 rows=136 loops=1)
         Hash Cond: ("outer".s_nationkey = "inner".n_nationkey)
         ->  Nested Loop  (cost=7493.08..7499.16 rows=2 width=62) (actual
time=1422.161..1486.113 rows=3520 loops=1)
               ->  Subquery Scan "IN_subquery"  (cost=7493.08..7493.11 rows=2
width=4) (actual time=1422.030..1438.571 rows=3520 loops=1)
                     ->  Unique  (cost=7493.08..7493.09 rows=2 width=4) (actual
time=1422.025..1432.330 rows=3520 loops=1)
                           ->  Sort  (cost=7493.08..7493.09 rows=2 width=4)
(actual time=1422.021..1424.966 rows=4330 loops=1)
```

```
                                Sort Key: partsupp.ps_suppkey
                                ->  Nested Loop  (cost=0.00..7493.07 rows=2
width=4) (actual time=0.410..1413.364 rows=4330 loops=1)
                                      ->  Seq Scan on part  (cost=0.00..7363.00
rows=1 width=4) (actual time=0.086..209.181 rows=1627 loops=1)
                                            Filter: ((p_name)::text ~~
'violet%'::text)
                                      ->  Index Scan using i_ps_partkey on
partsupp  (cost=0.00..129.98 rows=7 width=8) (actual time=0.306..0.730 rows=3
loops=1627)
                                            Index Cond: (partsupp.ps_partkey =
"outer".p_partkey)
                                            Filter: ((ps_availqty)::double
precision > (subplan))
                                            SubPlan
                                              ->  Aggregate  (cost=6.03..6.03
rows=1 width=4) (actual time=0.164..0.164 rows=1 loops=6508)
                                                    ->  Index Scan using
i_l_suppkey_partkey on lineitem  (cost=0.00..6.02 rows=1 width=4) (actual
time=0.100..0.157 rows=1 loops=6508)
                                                          Index Cond:
((l_partkey = $0) AND (l_suppkey = $1))
                                                          Filter: ((l_shipdate
>= '1996-01-01'::date) AND (l_shipdate < '1997-01-01 00:00:00'::timestamp without
time zone))
                ->  Index Scan using supplier_pkey on supplier  (cost=0.00..3.01
rows=1 width=66) (actual time=0.007..0.009 rows=1 loops=3520)
                      Index Cond: (supplier.s_suppkey = "outer".ps_suppkey)
          ->  Hash  (cost=1.31..1.31 rows=1 width=4) (actual time=0.089..0.089
rows=0 loops=1)
                ->  Seq Scan on nation  (cost=0.00..1.31 rows=1 width=4) (actual
time=0.083..0.085 rows=1 loops=1)
                      Filter: (n_name = 'UNITED STATES'::bpchar)
 Total runtime: 1492.060 ms
(26 rows)
```

PostgreSQL 7.4 と同様にインデックスを設定してみた。

```
create index i_p_name on part(p_name);
```

その結果、改善が見られた(リスト 2.20-5)。

リスト 2.20-5　PostgreSQL8.0 における改善後の問い合わせ計画

```
Sort  (cost=143.50..143.50 rows=1 width=58) (actual time=1324.329..1324.412
rows=136 loops=1)
   Sort Key: supplier.s_name
   ->  Hash Join  (cost=137.38..143.49 rows=1 width=58) (actual
time=1258.118..1324.076 rows=136 loops=1)
         Hash Cond: ("outer".s_nationkey = "inner".n_nationkey)
         ->  Nested Loop  (cost=136.07..142.15 rows=2 width=62) (actual
time=1257.706..1319.844 rows=3520 loops=1)
               ->  Subquery Scan "IN_subquery"  (cost=136.07..136.10 rows=2
width=4) (actual time=1257.579..1273.652 rows=3520 loops=1)
                     ->  Unique  (cost=136.07..136.08 rows=2 width=4) (actual
time=1257.574..1267.449 rows=3520 loops=1)
                           ->  Sort  (cost=136.07..136.07 rows=2 width=4) (actual
time=1257.571..1260.501 rows=4330 loops=1)
                                 Sort Key: partsupp.ps_suppkey
                                 ->  Nested Loop  (cost=0.00..136.06 rows=2
width=4) (actual time=0.451..1248.646 rows=4330 loops=1)
                                       ->  Index Scan using i_p_name on part
(cost=0.00..6.01 rows=1 width=4) (actual time=0.119..49.175 rows=1627 loops=1)
                                             Index Cond: (((p_name)::text >=
'violet'::character varying) AND ((p_name)::text < 'violeu'::character varying))
                                             Filter: ((p_name)::text ~~
'violet%'::text)
                                       ->  Index Scan using i_ps_partkey on
partsupp  (cost=0.00..129.96 rows=7 width=8) (actual time=0.308..0.726 rows=3
loops=1627)
                                             Index Cond: (partsupp.ps_partkey =
"outer".p_partkey)
                                             Filter: ((ps_availqty)::double
precision > (subplan))
                                             SubPlan
                                               ->  Aggregate  (cost=6.02..6.03
rows=1 width=4) (actual time=0.161..0.162 rows=1 loops=6508)
                                                     ->  Index Scan using
```

```
i_l_suppkey_partkey on lineitem  (cost=0.00..6.02 rows=1 width=4) (actual
time=0.098..0.155 rows=1 loops=6508)
                                                    Index Cond:
((l_partkey = $0) AND (l_suppkey = $1))
                                                    Filter: ((l_shipdate
>= '1996-01-01'::date) AND (l_shipdate < '1997-01-01 00:00:00'::timestamp without
time zone))
              ->  Index Scan using supplier_pkey on supplier  (cost=0.00..3.01
rows=1 width=66) (actual time=0.007..0.009 rows=1 loops=3520)
                    Index Cond: (supplier.s_suppkey = "outer".ps_suppkey)
        ->  Hash  (cost=1.31..1.31 rows=1 width=4) (actual time=0.087..0.087
rows=0 loops=1)
              ->  Seq Scan on nation  (cost=0.00..1.31 rows=1 width=4) (actual
time=0.081..0.082 rows=1 loops=1)
                    Filter: (n_name = 'UNITED STATES'::bpchar)
 Total runtime: 1325.400 ms
(27 rows)
```

2.21　Q21

問い合わせをリスト 2.21-1 に示す。

リスト 2.21-1　Q19 の問い合わせ

```
select s_name, count(*) as numwait from supplier, lineitem l1, orders, nation where
s_suppkey = l1.l_suppkey and o_orderkey = l1.l_orderkey and o_orderstatus = 'F' and
l1.l_receiptdate > l1.l_commitdate and exists ( select * from lineitem l2 where
l2.l_orderkey = l1.l_orderkey and l2.l_suppkey <> l1.l_suppkey ) and not exists
( select * from lineitem l3 where l3.l_orderkey = l1.l_orderkey and l3.l_suppkey <>
l1.l_suppkey and l3.l_receiptdate > l3.l_commitdate ) and s_nationkey = n_nationkey
and n_name = 'ROMANIA' group by s_name order by numwait desc, s_name LIMIT 100;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.21-2 に示す。

リスト 2.21-2　Q20 の実行計画

```
Limit  (cost=3109815.94..3109816.19 rows=100 width=29) (actual
time=15763.853..15764.038 rows=100 loops=1)
```

```
   ->  Sort  (cost=3109815.94..3109841.04 rows=10038 width=29) (actual
time=15763.849..15763.913 rows=100 loops=1)
         Sort Key: count(*), supplier.s_name
         ->  GroupAggregate  (cost=3108993.25..3109148.76 rows=10038 width=29)
(actual time=15754.182..15762.753 rows=398 loops=1)
               ->  Sort  (cost=3108993.25..3109036.72 rows=17388 width=29) (actual
time=15754.128..15756.658 rows=4031 loops=1)
                     Sort Key: supplier.s_name
                     ->  Merge Join  (cost=3052266.11..3107768.63 rows=17388
width=29) (actual time=12072.339..15738.067 rows=4031 loops=1)
                           Merge Cond: ("outer".o_orderkey = "inner".l_orderkey)
                           ->  Index Scan using orders_pkey on orders
(cost=0.00..53357.62 rows=710562 width=4) (actual time=0.129..2918.535 rows=729338
loops=1)
                                 Filter: (o_orderstatus = 'F'::bpchar)
                           ->  Sort  (cost=3052266.11..3052366.17 rows=40024
width=33) (actual time=12070.025..12082.761 rows=8237 loops=1)
                                 Sort Key: l1.l_orderkey
                                 ->  Nested Loop  (cost=1.32..3048593.15
rows=40024 width=33) (actual time=9.858..12010.311 rows=8237 loops=1)
                                       ->  Hash Join  (cost=1.32..400.93 rows=804
width=33) (actual time=0.103..27.958 rows=398 loops=1)
                                             Hash Cond: ("outer".s_nationkey =
"inner".n_nationkey)
                                             ->  Seq Scan on supplier
(cost=0.00..341.38 rows=10038 width=37) (actual time=0.005..15.656 rows=10000
loops=1)
                                             ->  Hash  (cost=1.31..1.31 rows=2
width=4) (actual time=0.050..0.050 rows=0 loops=1)
                                                   ->  Seq Scan on nation
(cost=0.00..1.31 rows=2 width=4) (actual time=0.041..0.046 rows=1 loops=1)
                                                         Filter: (n_name =
'ROMANIA'::bpchar)
                                       ->  Index Scan using i_l_suppkey on lineitem
l1  (cost=0.00..3790.66 rows=50 width=8) (actual time=2.936..29.970 rows=21
loops=398)
                                             Index Cond: ("outer".s_suppkey =
l1.l_suppkey)
                                             Filter: ((l_receiptdate >
l_commitdate) AND (subplan) AND (NOT (subplan)))
```

```
                                        SubPlan
                                          ->  Index Scan using i_l_orderkey
on lineitem l3  (cost=0.00..3.07 rows=2 width=127) (actual time=0.010..0.010 rows=1
loops=145337)
                                                Index Cond: (l_orderkey = $0)
                                                Filter: ((l_suppkey <> $1)
AND (l_receiptdate > l_commitdate))
                                        ->  Index Scan using i_l_orderkey
on lineitem l2  (cost=0.00..3.06 rows=4 width=127) (actual time=0.028..0.028 rows=1
loops=150765)
                                                Index Cond: (l_orderkey = $0)
                                                Filter: (l_suppkey <> $1)
 Total runtime: 15766.343 ms
(30 rows)
```

この問い合わせ計画では、特に改善の余地は見られなかった(sort_mem を増やしても改善はなかった)。

PostgreSQL 8.0 での問い合わせ計画をリスト 2.21-3 に示す。

リスト 2.21-3　PostgreSQL8.0 での問い合わせ計画

```
Limit  (cost=1143232.05..1143232.30 rows=100 width=29) (actual
time=9432.271..9432.451 rows=100 loops=1)
   ->  Sort  (cost=1143232.05..1143256.54 rows=9796 width=29) (actual
time=9432.268..9432.326 rows=100 loops=1)
         Sort Key: count(*), supplier.s_name
         ->  GroupAggregate  (cost=1142484.72..1142582.68 rows=9796 width=29)
(actual time=9423.461..9431.317 rows=398 loops=1)
               ->  Sort  (cost=1142484.72..1142509.21 rows=9796 width=29) (actual
time=9423.410..9425.938 rows=4033 loops=1)
                     Sort Key: supplier.s_name
                     ->  Nested Loop  (cost=1.31..1141835.34 rows=9796 width=29)
(actual time=0.961..9408.209 rows=4033 loops=1)
                           ->  Nested Loop  (cost=1.31..1081250.20 rows=20005
width=33) (actual time=0.880..9006.578 rows=8241 loops=1)
                                 ->  Hash Join  (cost=1.31..401.31 rows=400
width=33) (actual time=0.169..27.321 rows=398 loops=1)
                                       Hash Cond: ("outer".s_nationkey =
"inner".n_nationkey)
```

```
                                    ->  Seq Scan on supplier
(cost=0.00..346.00 rows=10000 width=37) (actual time=0.026..14.948 rows=10000
loops=1)
                                    ->  Hash  (cost=1.31..1.31 rows=1 width=4)
(actual time=0.096..0.096 rows=0 loops=1)
                                          ->  Seq Scan on nation
(cost=0.00..1.31 rows=1 width=4) (actual time=0.087..0.091 rows=1 loops=1)
                                                Filter: (n_name =
'ROMANIA'::bpchar)
                              ->  Index Scan using i_l_suppkey on lineitem l1
(cost=0.00..2701.45 rows=54 width=8) (actual time=1.313..22.432 rows=21 loops=398)
                                    Index Cond: ("outer".s_suppkey =
l1.l_suppkey)
                                    Filter: ((l_receiptdate > l_commitdate) AND
(subplan) AND (NOT (subplan)))
                                    SubPlan
                                      ->  Index Scan using i_l_orderkey on
lineitem l3  (cost=0.00..9.31 rows=60 width=127) (actual time=0.010..0.010 rows=1
loops=145330)
                                            Index Cond: (l_orderkey = $0)
                                            Filter: ((l_suppkey <> $1) AND
(l_receiptdate > l_commitdate))
                                      ->  Index Scan using i_l_orderkey on
lineitem l2  (cost=0.00..8.86 rows=179 width=127) (actual time=0.021..0.021 rows=1
loops=150746)
                                            Index Cond: (l_orderkey = $0)
                                            Filter: (l_suppkey <> $1)
                         ->  Index Scan using orders_pkey on orders
(cost=0.00..3.02 rows=1 width=4) (actual time=0.043..0.043 rows=0 loops=8241)
                              Index Cond: (orders.o_orderkey =
"outer".l_orderkey)
                              Filter: (o_orderstatus = 'F'::bpchar)
 Total runtime: 9433.757 ms
(28 rows)
```

こちらも特に改善の余地はなかった。

2.22 Q22

問い合わせをリスト 2.22-1 に示す。

リスト 2.22-1　Q22 の問い合わせ

```
select cntrycode, count(*) as numcust, sum(c_acctbal) as totacctbal from ( select
substr(c_phone, 1, 2) as cntrycode, c_acctbal from customer where substr(c_phone,
1, 2) in ('34', '16', '19', '23', '24', '30', '22') and c_acctbal > ( select
avg(c_acctbal) from customer where c_acctbal > 0.00 and substr(c_phone, 1, 2) in
('34', '16', '19', '23', '24', '30', '22') ) and not exists ( select * from orders
where o_custkey = c_custkey ) ) as vip group by cntrycode order by cntrycode;
```

この問い合わせに対する実行計画を EXPLAIN ANALYZE コマンドで出力したものをリスト 2.22-2 に示す。

リスト 2.22-2　Q22 の実行計画

```
GroupAggregate  (cost=643782.91..643800.17 rows=863 width=23) (actual
time=3747.325..3759.849 rows=7 loops=1)
   InitPlan
     ->  Aggregate  (cost=13722.56..13722.56 rows=1 width=4) (actual
time=1543.448..1543.449 rows=1 loops=1)
           ->  Seq Scan on customer  (cost=0.00..13710.85 rows=4682 width=4)
(actual time=0.037..1488.599 rows=38317 loops=1)
                 Filter: ((c_acctbal > 0::double precision) AND
((substr((c_phone)::text, 1, 2) = '34'::text) OR (substr((c_phone)::text, 1, 2) =
'16'::text) OR (substr((c_phone)::text, 1, 2) = '19'::text) OR
(substr((c_phone)::text, 1, 2) = '23'::text) OR (substr((c_phone)::text, 1, 2) =
'24'::text) OR (substr((c_phone)::text, 1, 2) = '30'::text) OR
(substr((c_phone)::text, 1, 2) = '22'::text)))
   ->  Sort  (cost=630060.36..630062.51 rows=863 width=23) (actual
time=3745.294..3749.329 rows=6400 loops=1)
         Sort Key: substr((customer.c_phone)::text, 1, 2)
         ->  Seq Scan on customer  (cost=0.00..630018.27 rows=863 width=23) (actual
time=1543.777..3723.209 rows=6400 loops=1)
               Filter: (((substr((c_phone)::text, 1, 2) = '34'::text) OR
(substr((c_phone)::text, 1, 2) = '16'::text) OR (substr((c_phone)::text, 1, 2) =
'19'::text) OR (substr((c_phone)::text, 1, 2) = '23'::text) OR
(substr((c_phone)::text, 1, 2) = '24'::text) OR (substr((c_phone)::text, 1, 2) =
'30'::text) OR (substr((c_phone)::text, 1, 2) = '22'::text)) AND (c_acctbal > $0)
AND (NOT (subplan)))
```

```
              SubPlan
                ->  Index Scan using i_o_custkey on orders  (cost=0.00..69.81
rows=17 width=115) (actual time=0.030..0.030 rows=1 loops=19140)
                      Index Cond: (o_custkey = $1)
 Total runtime: 3761.464 ms
(13 rows)
```

この問い合わせでは、substr 関数が多用されており、この部分を関数インデックスにすることで事前に計算を行うと共に、インデックスが使用されることによって高速化の効果が期待できる。

```
create index i_substr_phone_index on customer((substr(c_phone, 1, 2)));
```

ただし、PostgreSQL 7.4 では関数インデックスに対する統計情報は正しく作られないので、このままではインデックスが使用されない。そこで、以下の設定を併用する。

```
set enable_seqscan to off;
```

結果をリスト 2.22-3 に示す。

リスト 2.22-3　改善後の問い合わせ計画

```
GroupAggregate  (cost=60062.47..60079.71 rows=862 width=23) (actual
time=1903.659..1914.926 rows=7 loops=1)
   InitPlan
     ->  Aggregate  (cost=19567.44..19567.44 rows=1 width=4) (actual
time=611.623..611.624 rows=1 loops=1)
           ->  Index Scan using i_substr_phone_index, i_substr_phone_index,
i_substr_phone_index, i_substr_phone_index, i_substr_phone_index,
i_substr_phone_index, i_substr_phone_index on customer  (cost=0.00..19555.79
rows=4656 width=4) (actual time=0.014..547.608 rows=38317 loops=1)
                 Index Cond: ((substr((c_phone)::text, 1, 2) = '34'::text) OR
(substr((c_phone)::text, 1, 2) = '16'::text) OR (substr((c_phone)::text, 1, 2) =
'19'::text) OR (substr((c_phone)::text, 1, 2) = '23'::text) OR
(substr((c_phone)::text, 1, 2) = '24'::text) OR (substr((c_phone)::text, 1, 2) =
'30'::text) OR (substr((c_phone)::text, 1, 2) = '22'::text))
                 Filter: (c_acctbal > 0::real)
   ->  Sort  (cost=40495.03..40497.19 rows=862 width=23) (actual
time=1901.789..1905.683 rows=6400 loops=1)
```

```
        Sort Key: substr((customer.c_phone)::text, 1, 2)
        ->  Index Scan using i_substr_phone_index, i_substr_phone_index,
i_substr_phone_index, i_substr_phone_index, i_substr_phone_index,
i_substr_phone_index, i_substr_phone_index on customer  (cost=0.00..40453.00
rows=862 width=23) (actual time=612.641..1884.468 rows=6400 loops=1)
              Index Cond: ((substr((c_phone)::text, 1, 2) = '34'::text) OR
(substr((c_phone)::text, 1, 2) = '16'::text) OR (substr((c_phone)::text, 1, 2) =
'19'::text) OR (substr((c_phone)::text, 1, 2) = '23'::text) OR
(substr((c_phone)::text, 1, 2) = '24'::text) OR (substr((c_phone)::text, 1, 2) =
'30'::text) OR (substr((c_phone)::text, 1, 2) = '22'::text))
              Filter: ((c_acctbal > $0) AND (NOT (subplan)))
              SubPlan
                ->  Index Scan using i_o_custkey on orders  (cost=0.00..63.67
rows=16 width=115) (actual time=0.032..0.032 rows=1 loops=19140)
                      Index Cond: (o_custkey = $1)
 Total runtime: 1917.389 ms
(15 rows)
```

PostgreSQL 8.0 での問い合わせ計画をリスト 2.22-4 に示す。

リスト 2.22-4　PostgreSQL8.0 での問い合わせ計画

```
GroupAggregate  (cost=645442.78..645460.02 rows=862 width=23) (actual
time=3314.736..3326.906 rows=7 loops=1)
   InitPlan
     ->  Aggregate  (cost=13791.79..13791.79 rows=1 width=4) (actual
time=1427.940..1427.941 rows=1 loops=1)
           ->  Seq Scan on customer  (cost=0.00..13780.00 rows=4713 width=4)
(actual time=0.034..1374.779 rows=38317 loops=1)
                 Filter: ((c_acctbal > 0::double precision) AND
((substr((c_phone)::text, 1, 2) = '34'::text) OR (substr((c_phone)::text, 1, 2) =
'16'::text) OR (substr((c_phone)::text, 1, 2) = '19'::text) OR
(substr((c_phone)::text, 1, 2) = '23'::text) OR (substr((c_phone)::text, 1, 2) =
'24'::text) OR (substr((c_phone)::text, 1, 2) = '30'::text) OR
(substr((c_phone)::text, 1, 2) = '22'::text)))
   ->  Sort  (cost=631651.00..631653.15 rows=862 width=23) (actual
time=3312.766..3316.780 rows=6400 loops=1)
         Sort Key: substr((customer.c_phone)::text, 1, 2)
         ->  Seq Scan on customer  (cost=0.00..631608.97 rows=862 width=23) (actual
```

```
time=1428.272..3290.550 rows=6400 loops=1)
              Filter: (((substr((c_phone)::text, 1, 2) = '34'::text) OR
(substr((c_phone)::text, 1, 2) = '16'::text) OR (substr((c_phone)::text, 1, 2) =
'19'::text) OR (substr((c_phone)::text, 1, 2) = '23'::text) OR
(substr((c_phone)::text, 1, 2) = '24'::text) OR (substr((c_phone)::text, 1, 2) =
'30'::text) OR (substr((c_phone)::text, 1, 2) = '22'::text)) AND (c_acctbal > $0)
AND (NOT (subplan)))
              SubPlan
                ->  Index Scan using i_o_custkey on orders  (cost=0.00..74.14
rows=18 width=114) (actual time=0.018..0.018 rows=1 loops=19140)
                      Index Cond: (o_custkey = $1)
 Total runtime: 3328.645 ms
(13 rows)
```

PostgreSQL 7.4 と同様に関数インデックスを設定する。PostgreSQL 8.0 では関数インデックスの統計情報も取得されるので、併せて統計情報も更新する。

```
create index i_substr_phone_index on customer((substr(c_phone, 1, 2)));
vacuum analyze customer;
```

結果をリスト 2.22-5 に示す。

リスト 2.22-5　PostgreSQL8.0 における関数インデックス設定後の問い合わせ計画

```
Sort  (cost=319324.46..319324.46 rows=2 width=23) (actual time=2588.437..2588.442
rows=7 loops=1)
   Sort Key: substr((customer.c_phone)::text, 1, 2)
   InitPlan
     ->  Aggregate  (cost=13864.86..13864.86 rows=1 width=4) (actual
time=1430.489..1430.490 rows=1 loops=1)
           ->  Seq Scan on customer  (cost=0.00..13780.00 rows=33943 width=4)
(actual time=0.145..1377.438 rows=38317 loops=1)
                 Filter: ((c_acctbal > 0::double precision) AND
((substr((c_phone)::text, 1, 2) = '34'::text) OR (substr((c_phone)::text, 1, 2) =
'16'::text) OR (substr((c_phone)::text, 1, 2) = '19'::text) OR
(substr((c_phone)::text, 1, 2) = '23'::text) OR (substr((c_phone)::text, 1, 2) =
'24'::text) OR (substr((c_phone)::text, 1, 2) = '30'::text) OR
(substr((c_phone)::text, 1, 2) = '22'::text)))
   ->  HashAggregate  (cost=305459.57..305459.59 rows=2 width=23) (actual
```

```
time=2588.404..2588.414 rows=7 loops=1)
        ->  Index Scan using i_substr_phone_index, i_substr_phone_index,
i_substr_phone_index, i_substr_phone_index, i_substr_phone_index,
i_substr_phone_index, i_substr_phone_index on customer  (cost=0.00..305412.96
rows=6214 width=23) (actual time=1430.783..2566.133 rows=6400 loops=1)
              Index Cond: ((substr((c_phone)::text, 1, 2) = '34'::text) OR
(substr((c_phone)::text, 1, 2) = '16'::text) OR (substr((c_phone)::text, 1, 2) =
'19'::text) OR (substr((c_phone)::text, 1, 2) = '23'::text) OR
(substr((c_phone)::text, 1, 2) = '24'::text) OR (substr((c_phone)::text, 1, 2) =
'30'::text) OR (substr((c_phone)::text, 1, 2) = '22'::text))
              Filter: ((c_acctbal > $0) AND (NOT (subplan)))
              SubPlan
                ->  Index Scan using i_o_custkey on orders  (cost=0.00..74.14
rows=18 width=114) (actual time=0.029..0.029 rows=1 loops=19140)
                      Index Cond: (o_custkey = $1)
 Total runtime: 2589.159 ms
(14 rows)
```

一部インデックスが使われていないところがあるので、

```
set enable_seqscan to off;
```

を実施する．最終的にはリスト 2.22-6 の結果を得た。

リスト 2.22-6　PostgreSQL8.0 における最終的な問い合わせ計画

```
Sort  (cost=437935.45..437935.45 rows=2 width=23) (actual time=1707.642..1707.648
rows=7 loops=1)
   Sort Key: substr((customer.c_phone)::text, 1, 2)
   InitPlan
     ->  Aggregate  (cost=132475.85..132475.85 rows=1 width=4) (actual
time=550.795..550.796 rows=1 loops=1)
           ->  Index Scan using i_substr_phone_index, i_substr_phone_index,
i_substr_phone_index, i_substr_phone_index, i_substr_phone_index,
i_substr_phone_index, i_substr_phone_index on customer  (cost=0.00..132390.99
rows=33943 width=4) (actual time=0.015..492.072 rows=38317 loops=1)
                 Index Cond: ((substr((c_phone)::text, 1, 2) = '34'::text) OR
(substr((c_phone)::text, 1, 2) = '16'::text) OR (substr((c_phone)::text, 1, 2) =
'19'::text) OR (substr((c_phone)::text, 1, 2) = '23'::text) OR
```

```
(substr((c_phone)::text, 1, 2) = '24'::text) OR (substr((c_phone)::text, 1, 2) =
'30'::text) OR (substr((c_phone)::text, 1, 2) = '22'::text))
                Filter: (c_acctbal > 0::double precision)
   ->  HashAggregate  (cost=305459.57..305459.59 rows=2 width=23) (actual
time=1707.607..1707.618 rows=7 loops=1)
         ->  Index Scan using i_substr_phone_index, i_substr_phone_index,
i_substr_phone_index, i_substr_phone_index, i_substr_phone_index,
i_substr_phone_index, i_substr_phone_index on customer  (cost=0.00..305412.96
rows=6214 width=23) (actual time=551.102..1685.543 rows=6400 loops=1)
               Index Cond: ((substr((c_phone)::text, 1, 2) = '34'::text) OR
(substr((c_phone)::text, 1, 2) = '16'::text) OR (substr((c_phone)::text, 1, 2) =
'19'::text) OR (substr((c_phone)::text, 1, 2) = '23'::text) OR
(substr((c_phone)::text, 1, 2) = '24'::text) OR (substr((c_phone)::text, 1, 2) =
'30'::text) OR (substr((c_phone)::text, 1, 2) = '22'::text))
               Filter: ((c_acctbal > $0) AND (NOT (subplan)))
               SubPlan
                 ->  Index Scan using i_o_custkey on orders  (cost=0.00..74.14
rows=18 width=114) (actual time=0.029..0.029 rows=1 loops=19140)
                       Index Cond: (o_custkey = $1)
 Total runtime: 1709.382 ms
(15 rows)
```

3 まとめ

インデックスの設定、ソートメモリなどのパラメータの設定により、DBT-3 のいくつかの問い合わせの実行時間を改善した。

3.1 実行時間の改善率

その結果を、表 3.1-1 と表 3.1-2 にサマライズする。なお、改善のなかったものについては「主な改善内容」を「－」で示す。「実行時間」は、(改善後の実行時間/改善前の実行時間)を%で示した。

表 3.1-1　PostgreSQL7.4

問い合わせ番号	主な改善内容	実行時間
Q1	－	
Q2	インデックスの設定	79%
Q3	ソートメモリの設定	76%
Q4	－	
Q5	－	
Q6	－	
Q7	－	
Q8	ソートメモリ、インデックスの設定	54%
Q9	－	
Q10	ソートメモリ、インデックスの設定	35%
Q11	－	
Q12	－	
Q13	－	
Q14	インデックスの設定	37%
Q15	－	
Q16	ソートメモリの設定	81%
Q17	インデックスの設定	61%
Q18	－	
Q19	インデックスの設定	70%
Q20	インデックスの設定	91%
Q21	－	
Q22	インデックスの設定	50%

表 3.1-2　PostgreSQL8.0

問い合わせ番号	主な改善内容	実行時間
Q1	―	
Q2	インデックスの設定	73%
Q3	ソートメモリの設定	70%
Q4	―	
Q5	―	
Q6	インデックスの設定	15%
Q7	―	
Q8	ソートメモリ、インデックスの設定	63%
Q9	―	
Q10	ソートメモリの設定	79%
Q11	―	
Q12	―	
Q13	―	
Q14	インデックスの設定	35%
Q15	―	
Q16	ソートメモリの設定	70%
Q17	インデックスの設定	54%
Q18	―	
Q19	―	
Q20	インデックスの設定	88%
Q21	―	
Q22	インデックスの設定	51%

3.2 設定したインデックス

今回のチューニングでは、性能改善のために以下のインデックスを追加した。

```
create index i_p_size on part(p_size);
create index i_l_shipdate_discount_quantity on
lineitem(l_shipdate,l_discount,l_quantity);
create index i_p_type on part(p_type);
create index i_p_brand_container on part(p_brand, p_container);
create index i_l_shipmode on lineitem(l_shipmode);
create index i_p_name on part(p_name);
create index i_substr_phone_index on customer((substr(c_phone, 1, 2)));
```

4 テーブルスペースの利用

PostgreSQL 8.0 では、テーブルスペース(tablespaces)が利用できる。テーブルスペースはテーブルやインデックスを物理的に別々のディスクドライブに配置することを可能にし、結果としてデータベースの問い合わせの実行速度を向上させるものである。

ここでは、DBT-3 の問い合わせ「Q3」を使ってテーブルスペースの効果を検証する。リスト 3.2-1 に問い合わせを再掲する。

リスト 3.2-1　Q3 の問い合わせ

```
select l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue,
o_orderdate, o_shippriority from customer, orders, lineitem where
c_mktsegment = 'BUILDING' and c_custkey = o_custkey and l_orderkey =
o_orderkey and o_orderdate < date '1995-03-11' and l_shipdate > date
'1995-03-11' group by l_orderkey, o_orderdate, o_shippriority order by
revenue desc, o_orderdate LIMIT 10;
```

この問い合わせでは、lineitem(600 万件)、orders(150 万件)、customer(15 万件)という、比較的大きなテーブルに順アクセスする(件数はスケールファクタ 1 のときの値)。そのため、ディスク I/O が律速になっており、テーブルスペースの効果が検証しやすいと考えられる。

今回検証に使ったシステムでは、/dbt3_0/pgsql にデータベースクラスタおよびトランザクションログを格納している。また、テーブルスペース用に 3 つのディスクドライブを用意した(表 3.2-1)。

表 3.2-1　今回使ったディスクドライブの配置

```
/dev/cciss/c0d2p1     70009904   6242644  60210932  10% /dbt3_0
/dev/cciss/c0d3p1     35002928     32828  33192040   1% /dbt3_1
/dev/cciss/c0d4p1     70009904     32828  66420748   1% /dbt3_2
/dev/cciss/c0d5p1     70009904     32828  66420748   1% /dbt3_3
```

この試験機上に以下のようにテーブルスペースを構築した。

```
# mkdir /dbt3_1/pgsql
# chown pgsql /dbt3_1/pgsql
# mkdir /dbt3_2/pgsql
# chown pgsql /dbt3_2/pgsql
# mkdir /dbt3_3/pgsql
```

```
# chown pgsql /dbt3_3/pgsql

DBT3=# create tablespace dbt3_1 location '/dbt3_1/pgsql';
CREATE TABLESPACE
DBT3=# create tablespace dbt3_2 location '/dbt3_2/pgsql';
CREATE TABLESPACE
DBT3=# create tablespace dbt3_3 location '/dbt3_3/pgsql';
CREATE TABLESPACE
```

そしてこれらのテーブルスペースに lineitem、orders、customer テーブルを配置する。

```
DBT3=# alter table lineitem set tablespace dbt3_1;
DBT3=# alter table orders set tablespace dbt3_2;
DBT3=# alter table customer set tablespace dbt3_3;
```

カーネルのバッファキャッシュの影響をなくすために、postmaster を一度停止した後、/dbt3_0、/dbt3_1、/dbt3_2、/dbt3_3 を umount/mount した。

この状態で問い合わせを実行したところ、リスト 3.2-2 の結果を得た。

リスト 3.2-2　テーブルスペースを使った実行結果

```
EXPLAIN ANALYZE select l_orderkey, sum(l_extendedprice * (1 - l_discount)) as
revenue, o_orderdate, o_shippriority from customer, orders, lineitem where
c_mktsegment = 'BUILDING' and c_custkey = o_custkey and l_orderkey = o_orderkey and
o_orderdate < date '1995-03-11' and l_shipdate > date '1995-03-11' group by
l_orderkey, o_orderdate, o_shippriority order by revenue desc, o_orderdate LIMIT 10;
```

```
Limit  (cost=420820.75..420820.78 rows=10 width=20) (actual
time=35999.099..35999.122 rows=10 loops=1)
   ->  Sort  (cost=420820.75..421604.92 rows=313665 width=20) (actual
time=35999.093..35999.101 rows=10 loops=1)
         Sort Key: sum((lineitem.l_extendedprice * (1::double precision -
lineitem.l_discount))), orders.o_orderdate
         ->  GroupAggregate  (cost=370531.62..376804.92 rows=313665 width=20)
(actual time=35834.564..35927.014 rows=11613 loops=1)
               ->  Sort  (cost=370531.62..371315.78 rows=313665 width=20) (actual
time=35834.470..35863.648 rows=30511 loops=1)
                     Sort Key: lineitem.l_orderkey, orders.o_orderdate,
```

```
orders.o_shippriority
                    ->  Merge Join  (cost=88980.97..326515.78 rows=313665
width=20) (actual time=11473.746..35678.011 rows=30511 loops=1)
                          Merge Cond: ("outer".l_orderkey = "inner".o_orderkey)
                          ->  Index Scan using i_l_orderkey on lineitem
(cost=0.00..224757.40 rows=3229034 width=12) (actual time=24.823..20809.074
rows=3251844 loops=1)
                                Filter: (l_shipdate > '1995-03-11'::date)
                          ->  Sort  (cost=88980.97..89345.24 rows=145708
width=12) (actual time=11445.585..11641.525 rows=165481 loops=1)
                                Sort Key: orders.o_orderkey
                                ->  Hash Join  (cost=6096.75..74519.11
rows=145708 width=12) (actual time=568.637..10671.225 rows=146583 loops=1)
                                      Hash Cond: ("outer".o_custkey =
"inner".c_custkey)
                                      ->  Seq Scan on orders
(cost=0.00..48030.00 rows=730952 width=16) (actual time=0.546..4761.384 rows=724759
loops=1)
                                            Filter: (o_orderdate <
'1995-03-11'::date)
                                      ->  Hash  (cost=5905.00..5905.00
rows=29901 width=4) (actual time=567.688..567.688 rows=0 loops=1)
                                            ->  Seq Scan on customer
(cost=0.00..5905.00 rows=29901 width=4) (actual time=0.802..526.858 rows=30142
loops=1)
                                                  Filter: (c_mktsegment =
'BUILDING'::bpchar)
 Total runtime: 36038.827 ms
(20 rows)
```

一方、テーブルスペースを使わず、すべてのファイルを/dbt3_0 に置いたもので、/dbt3_0 を umount/mount した場合にはリスト 3.2-3 の結果を得た。

リスト 3.2-3　テーブルスペースを使わない実行結果

```
Limit  (cost=420820.75..420820.78 rows=10 width=20) (actual
time=46591.570..46591.592 rows=10 loops=1)
   ->  Sort  (cost=420820.75..421604.92 rows=313665 width=20) (actual
time=46591.565..46591.575 rows=10 loops=1)
```

```
        Sort Key: sum((lineitem.l_extendedprice * (1::double precision -
lineitem.l_discount))), orders.o_orderdate
        ->  GroupAggregate  (cost=370531.62..376804.92 rows=313665 width=20)
(actual time=46431.822..46521.399 rows=11613 loops=1)
              ->  Sort  (cost=370531.62..371315.78 rows=313665 width=20) (actual
time=46431.735..46461.063 rows=30511 loops=1)
                    Sort Key: lineitem.l_orderkey, orders.o_orderdate,
orders.o_shippriority
                    ->  Merge Join  (cost=88980.97..326515.78 rows=313665
width=20) (actual time=11675.381..46274.107 rows=30511 loops=1)
                          Merge Cond: ("outer".l_orderkey = "inner".o_orderkey)
                          ->  Index Scan using i_l_orderkey on lineitem
(cost=0.00..224757.40 rows=3229034 width=12) (actual time=9.710..31259.209
rows=3251844 loops=1)
                                Filter: (l_shipdate > '1995-03-11'::date)
                          ->  Sort  (cost=88980.97..89345.24 rows=145708
width=12) (actual time=11657.339..11854.033 rows=165481 loops=1)
                                Sort Key: orders.o_orderkey
                                ->  Hash Join  (cost=6096.75..74519.11
rows=145708 width=12) (actual time=584.254..10950.035 rows=146583 loops=1)
                                      Hash Cond: ("outer".o_custkey =
"inner".c_custkey)
                                      ->  Seq Scan on orders
(cost=0.00..48030.00 rows=730952 width=16) (actual time=3.515..5110.377 rows=724759
loops=1)
                                            Filter: (o_orderdate <
'1995-03-11'::date)
                                      ->  Hash  (cost=5905.00..5905.00
rows=29901 width=4) (actual time=580.349..580.349 rows=0 loops=1)
                                            ->  Seq Scan on customer
(cost=0.00..5905.00 rows=29901 width=4) (actual time=2.786..533.707 rows=30142
loops=1)
                                                  Filter: (c_mktsegment =
'BUILDING'::bpchar)
 Total runtime: 46609.616 ms
(20 rows)
```

このように、問い合わせ計画のコストは同様であるにも関わらず、テーブルスペースを使用した場合、実行時間は 46609.616 ms から 36038.827 ms へと短縮されている. 比率で言えば、77.3%に短縮されており、テーブルスペースの効果が確認できた。

このような結果となった理由の一つとしては、1 台のドライブで処理している場合には異なるファイルをアクセスする度にディスクヘッドが動くのに対し、テーブルスペースによってそれぞれのテーブルファイルにディスクドライブが割り当てられている場合はヘッドのシークがほとんど起きないためであると考えられる。

なお、ディスクドライブを umount/mount しない場合は、カーネルのバッファキャッシュの効果により、テーブルスペースの効果はこれほどはっきりとは確認できなかった。しかし、負荷の高い実環境や、より大容量のデータベースにおいては、バッファキャッシュの効果が薄れるため、テーブルスペースを利用するメリットがあると考えられる。

参考までに、今回使用したディスクドライブの性能を hdparam で測定した結果を表 3.2-2 に示す。

表 3.2-2　hdparam によるディスク速度の測定

```
/dev/cciss/c0d2p1:
 Timing buffered disk reads:  218 MB in  3.02 seconds =  72.19 MB/sec
/dev/cciss/c0d3p1:
 Timing buffered disk reads:  226 MB in  3.00 seconds =  75.33 MB/sec
/dev/cciss/c0d4p1:
 Timing buffered disk reads:  226 MB in  3.01 seconds =  75.08 MB/sec
/dev/cciss/c0d5p1:
 Timing buffered disk reads:  226 MB in  3.01 seconds =  75.08 MB/se
```