PHYSICAL DATABASE

ASSIGNMENT TWO

QUERY OPTIMIZATION

PEIFEN LU 18008550

1. Baseline Setup

Create a complex Join Query with sub-query as a baseline query:

```
SELECT c.custid,s.custid,p.prodid,c.first_name,c.surname,s.sellprice,c.address,s.qty,p.price
        FROM sales s, customer c, product p
        WHERE s.prodid=p.prodid AND s.custid=c.custid
        AND c.first_name = ANY (SELECT c.first_name FROM customer c
                WHERE c.first_name like 'C%' OR c.first_name like 'G%' OR c.first_name like 'F%'
                OR c.first_name like 'E%' OR c.first_name like 'J%')
        AND c.surname = ANY (SELECT c.surname FROM customer c
                WHERE c.surname like 'H%' OR c.surname like 'D%' OR c.surname like 'B%'
                OR c.surname like 'L%' OR c.surname like 'F%')
        AND s.sellprice > (SELECT min(s.sellprice) FROM sales s, product p
                WHERE s.prodid=p.prodid
                AND (p.pname like 'P%' OR p.pname like 'M%' OR p.pname like 'K%'))
        AND c.address = ANY (SELECT c.address FROM sales s, customer c
                WHERE s.custid= c.custid
                AND (c.address='AUCKLAND' OR c.address='WELLINGTON' OR c.address='NAPIER'))
        AND p.prodid = ANY (SELECT p.prodid FROM product p, sales s
                WHERE s.prodid= p.prodid AND p.price>2)
        AND s.qty = ANY (SELECT s.qty FROM sales s, product p
                WHERE s.prodid=p.prodid
                AND (p.pdesc like 'D2%' OR p.pdesc like 'D3%' OR p.pdesc like 'D5%'
                OR p.pdesc like 'D8%' OR p.pdesc like 'D7%'))
        AND p.pdesc = ANY (SELECT p.pdesc FROM product p
                WHERE p.pdesc like 'D2%' OR p.pdesc like 'D3%' OR p.pdesc like
                'D5%' OR p.pdesc like 'D8%' OR p.pdesc like 'D7%' OR p.pdesc like 'D9%' OR
                p.pdesc like 'D1%')
        AND p.price > (SELECT min(p.price) FROM sales s, product p
                WHERE s.prodid=p.prodid)
        ORDER BY p.prodid,s.saleid,c.custid, c.address
```

Trace Result:

BASELINE Trace Result

```
********************************************************************************
count    = number of times OCI procedure was executed
cpu      = cpu time in seconds executing
elapsed  = elapsed time in seconds executing
disk     = number of physical reads of buffers from disk
query    = number of buffers gotten for consistent read
current  = number of buffers gotten in current mode (usually for update)
rows     = number of rows processed by the fetch or execute call
********************************************************************************
```

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        2      0.06       0.06          0         42          0           0
Execute      2      0.00       0.00          0          0          0           0
Fetch     1961      2.01       4.99      12530      10926          3       29384
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     1965      2.08       5.06      12530      10968          3       29384

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 10435

Rows     Row Source Operation
-------  ---------------------------------------------------
  29384  SORT ORDER BY (cr=7431 pr=12530 pw=12185 time=4952853 us)
  29384   HASH JOIN SEMI (cr=7431 pr=12249 pw=11904 time=4574043 us)
  29384    HASH JOIN SEMI (cr=6114 pr=10734 pw=10734 time=4249831 us)
  29985     HASH JOIN SEMI (cr=5195 pr=10374 pw=10374 time=4174425 us)
  29985      HASH JOIN SEMI (cr=4276 pr=9534 pw=9534 time=3797671 us)
 161286       HASH JOIN SEMI (cr=3869 pr=7896 pw=7896 time=3264936 us)
 171282        HASH JOIN SEMI (cr=3462 pr=6111 pw=6111 time=2557727 us)
 199271         HASH JOIN  (cr=3453 pr=3766 pw=3766 time=1746229 us)
 199907          HASH JOIN  (cr=2525 pr=1645 pw=1645 time=902440 us)
 199907           TABLE ACCESS FULL SALES (cr=1829 pr=469 pw=469 time=465543 us)
      1            SORT AGGREGATE (cr=919 pr=469 pw=469 time=265589 us)
  42234             HASH JOIN  (cr=919 pr=469 pw=469 time=280841 us)
 200000              TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=200016 us)
    300              TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=707 us)
  60000           TABLE ACCESS FULL CUSTOMER (cr=696 pr=0 pw=0 time=60047 us)
    996          TABLE ACCESS FULL PRODUCT (cr=928 pr=280 pw=280 time=188278 us)
      1           SORT AGGREGATE (cr=919 pr=280 pw=280 time=188237 us)
 200000            HASH JOIN  (cr=919 pr=280 pw=280 time=150777 us)
 200000             TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=200021 us)
   1000             TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=23 us)
    800         TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=834 us)
  50000        TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=50049 us)
  30000       TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=30047 us)
  70488      VIEW  VW_NSO_3 (cr=919 pr=441 pw=441 time=394494 us)
  70488       HASH JOIN  (cr=919 pr=441 pw=441 time=323994 us)
 200000        TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=200030 us)
    500        TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=533 us)
 197303     VIEW  VW_NSO_2 (cr=919 pr=0 pw=0 time=198032 us)
 197303      HASH JOIN  (cr=919 pr=0 pw=0 time=727 us)
    990       TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=1019 us)
 200000       TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=18 us)
 190000    VIEW  VW_NSO_1 (cr=1317 pr=525 pw=525 time=286993 us)
 190000     HASH JOIN  (cr=1317 pr=525 pw=525 time=96992 us)
 200000      TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=34 us)
  50000      TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=99 us)
```

Baseline Time after running the query:
Count: 1965
CPU: 2.08
Elapsed: 5.06
Disk: 12530
Query: 10968
Current: 3
Rows: 29384

Explanation:

As we can see from the baseline time above, this query has a high CPU and elapsed time because the query has performed a full-table scan on all tables including the product table, sales table, and customer table. This query is a complex query with many joins and conditions. By doing a full table scan on every retrieve of that table, the performance becomes very poor. This query also has joint and sort operations which makes the query with poor performance. The total elapsed time is over 5 seconds and to read 29384 rows. The query opened too many buffers and read 10968 blocks to get the results, which results in poor performance.

2. Index Setup

Experiment #1:
I created several types of experiments using indexing to make the query run faster. These three tables are unstructured and do not contain any primary keys. The first experiment I did is to set up the primary keys and create indexes on the foreign keys to see whether it helps with the performance.

Creating Indexes Scripts:

```
--Experiment#1.
--set up the primary keys and create indexes on the foreign keys
alter table customer
add constraint customer_pk primary key(custid);

alter table product
add constraint product_pk primary key(prodid);

alter table sales
add constraint sales_pk primary key(saleid);

CREATE INDEX sales_custid_idex ON sales(custid);
CREATE INDEX sales_prodid_idex ON sales(prodid);
commit;

--show all indexes created
SELECT * from user_indexes;
```

Trace Result:

Trace Result

```
SELECT c.custid,s.custid,p.prodid,c.first_name,c.surname,s.sellprice,
    c.address,s.qty,p.price
    FROM sales s, customer c, product p
    WHERE s.prodid=p.prodid AND s.custid=c.custid
    AND c.first_name = ANY (SELECT c.first_name FROM customer c
        WHERE c.first_name like 'C%' OR c.first_name like 'G%' OR c.first_name like 'F%'
        OR c.first_name like 'E%' OR c.first_name like 'J%')
    AND c.surname = ANY (SELECT c.surname FROM customer c
        WHERE c.surname like 'H%' OR c.surname like 'D%' OR c.surname like 'B%'
        OR c.surname like 'L%' OR c.surname like 'F%')
    AND s.sellprice > (SELECT min(s.sellprice) FROM sales s, product p
        WHERE s.prodid=p.prodid
        AND (p.pname like 'P%' OR p.pname like 'M%' OR p.pname like 'K%'))
    AND c.address = ANY (SELECT c.address FROM sales s, customer c
        WHERE s.custid= c.custid
        AND (c.address='AUCKLAND' OR c.address='WELLINGTON' OR c.address='NAPIER'))
    AND p.prodid = ANY (SELECT p.prodid FROM product p, sales s
        WHERE s.prodid= p.prodid AND p.price>2)
    AND s.qty = ANY (SELECT s.qty FROM sales s, product p
        WHERE s.prodid=p.prodid
        AND (p.pdesc like 'D2%' OR p.pdesc like 'D3%' OR p.pdesc like 'D5%'
        OR p.pdesc like 'D8%' OR p.pdesc like 'D7%'))
    AND p.pdesc = ANY (SELECT p.pdesc FROM product p
        WHERE p.pdesc like 'D2%' OR p.pdesc like 'D3%' OR p.pdesc like
        'D5%' OR p.pdesc like 'D8%' OR p.pdesc like 'D7%' OR p.pdesc like 'D9%' OR
        p.pdesc like 'D1%')
    AND p.price > (SELECT min(p.price) FROM sales s, product p
        WHERE s.prodid=p.prodid)
    ORDER BY p.prodid,s.saleid,c.custid, c.address

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.16       0.16          0         77          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     1960      1.04       2.35       6552       5676          8       29384
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     1962      1.20       2.51       6552       5753          8       29384

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 10435

Rows     Row Source Operation
-------  ---------------------------------------------------
  29384  SORT ORDER BY (cr=5676 pr=6552 pw=5067 time=2345982 us)
  29384   HASH JOIN SEMI (cr=5676 pr=6268 pw=4783 time=2128117 us)
  29384    HASH JOIN SEMI (cr=4757 pr=5818 pw=4333 time=2009284 us)
  29384     HASH JOIN SEMI (cr=3918 pr=3720 pw=3311 time=1511487 us)
  29384      HASH JOIN SEMI (cr=3511 pr=3244 pw=2835 time=1311864 us)
 169362       HASH JOIN  (cr=3104 pr=1536 pw=1127 time=799984 us)
 169362        HASH JOIN  (cr=2697 pr=409 pw=0 time=321555 us)
    791         HASH JOIN SEMI (cr=868 pr=409 pw=0 time=116034 us)
    990          HASH JOIN SEMI (cr=859 pr=409 pw=0 time=106098 us)
    996           TABLE ACCESS FULL PRODUCT (cr=434 pr=409 pw=0 time=51657 us)
      1            SORT AGGREGATE (cr=425 pr=409 pw=0 time=51601 us)
 200000             HASH JOIN  (cr=425 pr=409 pw=0 time=800851 us)
   1000              TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=24 us)
 200000              INDEX FAST FULL SCAN SALES_PRODID_IDEX (cr=416 pr=409 pw=0 time=400456 us)(object id 732784)
 197303           VIEW  VW_NSO_2 (cr=425 pr=0 pw=0 time=197677 us)
 197303            HASH JOIN  (cr=425 pr=0 pw=0 time=373 us)
    990             TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=29 us)
 200000             INDEX FAST FULL SCAN SALES_PRODID_IDEX (cr=416 pr=0 pw=0 time=11 us)(object id 732784)
    800          TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=823 us)
 199907        TABLE ACCESS FULL SALES (cr=1829 pr=0 pw=0 time=236073 us)
      1         SORT AGGREGATE (cr=919 pr=0 pw=0 time=36146 us)
  42234          HASH JOIN  (cr=919 pr=0 pw=0 time=42545 us)
    300           TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=23 us)
 200000           TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=5 us)
  60000       TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=60032 us)
  30000      TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=30045 us)
  50000     TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=50036 us)
 190000    VIEW  VW_NSO_1 (cr=839 pr=775 pw=350 time=453702 us)
 190000     HASH JOIN  (cr=839 pr=775 pw=350 time=263698 us)
  50000      TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=50075 us)
 200000      INDEX FAST FULL SCAN SALES_CUSTID_IDEX (cr=432 pr=425 pw=0 time=200407 us)(object id 732783)
  70488   VIEW  VW_NSO_3 (cr=919 pr=0 pw=0 time=142228 us)
  70488    HASH JOIN  (cr=919 pr=0 pw=0 time=71738 us)
    500     TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=1045 us)
 200000     TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=21 us)
```

Explanation:

	Baseline	Experiment #1	% Change	Performance Result
Count	1965	1962	-0.15%	better than the baseline
CPU	2.08	1.20	-42%	better than the baseline
Elapsed	5.06	2.51	-50%	better than the baseline
Disk	12530	6552	-48%	better than the baseline
Query	10968	5753	-48%	better than the baseline
Current	3	8	167%	worse than the baseline
Rows	29384	29381	0%	Same as the baseline

% Change Formula: (Experiment #1 - Baseline)/Baseline

It has decreased the elapsed time by 50% and the CPU time by 42%. To read 29384 rows, the query opened only 5753 blocks, 48% less than the baseline. Compared with the baseline row source operation, the optimizer used some of the indexes for index fast full scan. However, as we can see from the trace file above, it did not use all the indexes I created.

Experiment #2:
To investigate further, I used hints to force the optimizer to use all the indexes I created.

Trace Result

```
SELECT /*+ INDEX(c customer_pk) INDEX(p product_pk) INDEX(s sales_pk
    sales_custid_idex sales_prodid_idex)*/
    c.custid,s.custid,p.prodid,
    c.first_name,c.surname,s.sellprice,c.address,s.qty,p.price
    FROM sales s, customer c, product p
    WHERE s.prodid=p.prodid AND s.custid=c.custid
    AND c.first_name = ANY (SELECT c.first_name FROM customer c
        WHERE c.first_name like 'C%' OR c.first_name like 'G%'
        OR c.first_name like 'F%' OR c.first_name like 'E%' OR c.first_name like 'J%')
    AND c.surname = ANY (SELECT c.surname FROM customer c
        WHERE c.surname like 'H%' OR c.surname like 'D%'
        OR c.surname like 'B%' OR c.surname like 'L%' OR c.surname like 'F%')
    AND s.sellprice > (SELECT min(s.sellprice) FROM sales s, product p
        WHERE s.prodid=p.prodid
        AND (p.pname like 'P%' OR p.pname like 'M%' OR
        p.pname like 'K%'))
    AND c.address = ANY (SELECT c.address FROM sales s, customer c
        WHERE s.custid= c.custid
        AND (c.address='AUCKLAND' OR c.address=
        'WELLINGTON' OR c.address='NAPIER'))
    AND p.prodid = ANY (SELECT p.prodid FROM product p, sales s
        WHERE s.prodid= p.prodid AND p.price>2)
    AND s.qty = ANY (SELECT s.qty FROM sales s, product p
        WHERE s.prodid=p.prodid
        AND (p.pdesc like 'D2%' OR p.pdesc like 'D3%' OR p.pdesc like 'D5%'
        OR p.pdesc like 'D8%' OR p.pdesc like 'D7%'))
    AND p.pdesc = ANY (SELECT p.pdesc FROM product p
        WHERE p.pdesc like 'D2%' OR p.pdesc like 'D3%' OR p.pdesc like 'D5%'
        OR p.pdesc like 'D8%' OR p.pdesc like 'D7%' OR p.pdesc like 'D9%' OR
        p.pdesc like 'D1%')
    AND p.price > (SELECT min(p.price) FROM sales s, product p
        WHERE s.prodid=p.prodid)
    ORDER BY p.prodid,s.saleid,c.custid, c.address
```

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.21       0.21          0         75          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     1960      1.18       2.61       6266       6222         10       29384
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     1962      1.40       2.82       6266       6297         10       29384

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 10435

Rows     Row Source Operation
-------  ---------------------------------------------------
  29384  SORT ORDER BY (cr=6222 pr=6266 pw=5072 time=2604927 us)
  29384   HASH JOIN SEMI (cr=6222 pr=5981 pw=4787 time=2427599 us)
  29384    HASH JOIN SEMI (cr=5303 pr=5576 pw=4382 time=2332195 us)
  29384     HASH JOIN SEMI (cr=4464 pr=3854 pw=3311 time=1977571 us)
  29384      HASH JOIN SEMI (cr=4057 pr=3378 pw=2835 time=1741248 us)
 169362       HASH JOIN  (cr=3650 pr=1670 pw=1127 time=1042320 us)
 169362        HASH JOIN  (cr=3119 pr=419 pw=0 time=331272 us)
    791         HASH JOIN SEMI (cr=875 pr=2 pw=0 time=125417 us)
    990          HASH JOIN SEMI (cr=866 pr=2 pw=0 time=116055 us)
    996           TABLE ACCESS BY INDEX ROWID PRODUCT (cr=441 pr=2 pw=0 time=49654 us)
   1000            INDEX FULL SCAN PRODUCT_PK (cr=3 pr=2 pw=0 time=392 us)(object id 732781)
      1            SORT AGGREGATE (cr=425 pr=0 pw=0 time=48249 us)
 200000             HASH JOIN  (cr=425 pr=0 pw=0 time=600438 us)
   1000              TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=33 us)
 200000              INDEX FAST FULL SCAN SALES_PRODID_IDEX (cr=416 pr=0 pw=0 time=200038 us)(object id 732784)
 197303           VIEW  VW_NSO_2 (cr=425 pr=0 pw=0 time=197757 us)
 197303            HASH JOIN  (cr=425 pr=0 pw=0 time=450 us)
    990             TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=57 us)
 200000             INDEX FAST FULL SCAN SALES_PRODID_IDEX (cr=416 pr=0 pw=0 time=12 us)(object id 732784)
    800         TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=20 us)
 199907        TABLE ACCESS BY INDEX ROWID SALES (cr=2244 pr=417 pw=0 time=236536 us)
 200000         INDEX FULL SCAN SALES_PK (cr=418 pr=417 pw=0 time=1521 us)(object id 732782)
      1         SORT AGGREGATE (cr=919 pr=0 pw=0 time=36537 us)
  42234          HASH JOIN  (cr=919 pr=0 pw=0 time=42535 us)
    300           TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=26 us)
 200000           TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=13 us)
  60000       TABLE ACCESS BY INDEX ROWID CUSTOMER (cr=531 pr=124 pw=0 time=240093 us)
  60000        INDEX FULL SCAN CUSTOMER_PK (cr=125 pr=124 pw=0 time=60583 us)(object id 732780)
  30000      TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=30050 us)
  50000     TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=50036 us)
 190000    VIEW  VW_NSO_1 (cr=839 pr=399 pw=399 time=251647 us)
 190000     HASH JOIN  (cr=839 pr=399 pw=399 time=61643 us)
  50000      TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=70 us)
 200000      INDEX FAST FULL SCAN SALES_CUSTID_IDEX (cr=432 pr=0 pw=0 time=36 us)(object id 732783)
  70488   VIEW  VW_NSO_3 (cr=919 pr=0 pw=0 time=71138 us)
  70488    HASH JOIN  (cr=919 pr=0 pw=0 time=71135 us)
    500     TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=31 us)
 200000     TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=11 us)
```

Explanation:

	Baseline	Experiment #2	% Change	Performance Result
Count	1965	1962	-0.15%	better than the baseline
CPU	2.08	1.40	-33%	better than the baseline
Elapsed	5.06	2.82	-44%	better than the baseline
Disk	12530	6266	-50%	better than the baseline
Query	10968	6297	-43%	better than the baseline
Current	3	10	233%	worse than the baseline
Rows	29384	29384	0%	Same as the baseline

% Change Formula: (Index Experiment #2 - Baseline)/Baseline

	Experiment #1	Experiment #2	% Change	Performance Result
Count	1962	1962	0%	Same as the Experiment #1
CPU	1.20	1.40	17%	Worse than the Experiment #1
Elapsed	2.51	2.82	12%	Worse than the Experiment #1
Disk	6552	6266	-4%	Better than the Experiment #1
Query	5753	6297	9.46%	Worse than the Experiment #1
Current	8	10	23%	Worse than the Experiment #1
Rows	29381	29384	0%	Same as the Experiment #1

% Change Formula: (Index Experiment #2 - Experiment #1)/ Experiment #1

As we can see from the trace results, the performance is better than the baseline but slightly worse than experiement#1. The query opened a reasonable number of buffers and read 6297 blocks to get the results, which is less than the baseline. The total elapsed time is 2.84 seconds and the total CPU is only 1.4 seconds. However, the parse time of these two is over 0.21 seconds, which is higher than when we did not use any hints. It means that the query is using unnecessary parse time. If we do not use hints, the optimizer compared all the plans and chose the one with the best performance it thinks. That’s why only part of the indexes I generated are used in experiment #1. The optimizer thinks it has the best performance.

Experiment #3:
The next experiment I designed is to create indexes on all join columns and predicate columns in where clause of the query to help with the performance.

Creating Indexes Scripts:

```
--Experiment#3.
/*set up more indexes on all join columns
and predicate columns in where clause of the query*/
CREATE INDEX customer_fname_idx ON customer(first_name);
CREATE INDEX customer_sname_idx ON customer(surname);
CREATE INDEX sales_qty_idx ON sales(qty);
CREATE BITMAP INDEX customer_address_idx ON customer(address);
CREATE INDEX sales_sellprice_idx ON sales(sellprice);
CREATE INDEX product_pname_idx ON product(pname);
CREATE INDEX product_price_idx ON product(price);
CREATE INDEX product_pdesc_idx ON product(pdesc);
commit;
```

Trace Result

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.25       0.26          0         86          0           0
Execute      1      0.06       0.06          0        425          0           0
Fetch     1960      1.15       2.57       6791     184255         14       29384
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     1962      1.47       2.89       6791     184766         14       29384

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 10435

Rows     Row Source Operation
-------  ---------------------------------------------------
  29384  SORT ORDER BY (cr=184680 pr=6791 pw=4998 time=2621147 us)
  29384   HASH JOIN SEMI (cr=184680 pr=6504 pw=4711 time=2439116 us)
  29384    HASH JOIN SEMI (cr=183761 pr=6119 pw=4326 time=2342142 us)
  29384     HASH JOIN SEMI (cr=182922 pr=4061 pw=3255 time=1903145 us)
  29384      HASH JOIN SEMI (cr=182722 pr=3448 pw=2835 time=1754810 us)
 169362       HASH JOIN  (cr=182522 pr=1547 pw=1127 time=1170194 us)
 169362        HASH JOIN  (cr=182115 pr=420 pw=0 time=506441 us)
    791         HASH JOIN SEMI (cr=1327 pr=4 pw=0 time=130239 us)
    990          HASH JOIN SEMI (cr=1320 pr=0 pw=0 time=117078 us)
    996           TABLE ACCESS BY INDEX ROWID PRODUCT (cr=895 pr=0 pw=0 time=63474 us)
    996            INDEX RANGE SCAN PRODUCT_PRICE_IDX (cr=428 pr=0 pw=0 time=61468 us)(object id 732791)
      1             SORT AGGREGATE (cr=425 pr=0 pw=0 time=61445 us)
 200000              HASH JOIN  (cr=425 pr=0 pw=0 time=601312 us)
   1000               TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=21 us)
 200000               INDEX FAST FULL SCAN SALES_PRODID_IDEX (cr=416 pr=0 pw=0 time=200040 us)(object id 732784)
 197303           VIEW  VW_NSO_2 (cr=425 pr=0 pw=0 time=197771 us)
 197303            HASH JOIN  (cr=425 pr=0 pw=0 time=467 us)
    990             TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=42 us)
 200000             INDEX FAST FULL SCAN SALES_PRODID_IDEX (cr=416 pr=0 pw=0 time=13 us)(object id 732784)
    800          INDEX FAST FULL SCAN PRODUCT_PDESC_IDX (cr=7 pr=4 pw=0 time=197 us)(object id 732792)
 199907         TABLE ACCESS BY INDEX ROWID SALES (cr=180788 pr=416 pw=0 time=637330 us)
 199907          INDEX RANGE SCAN SALES_SELLPRICE_IDX (cr=1336 pr=416 pw=0 time=38716 us)(object id 732789)
      1           SORT AGGREGATE (cr=919 pr=0 pw=0 time=37308 us)
  42234            HASH JOIN  (cr=919 pr=0 pw=0 time=42592 us)
    300             TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=329 us)
 200000             TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=14 us)
  60000        TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=48 us)
  30000       INDEX FAST FULL SCAN CUSTOMER_FNAME_IDX (cr=200 pr=193 pw=0 time=153850 us)(object id 732785)
  50000       INDEX FAST FULL SCAN CUSTOMER_SNAME_IDX (cr=200 pr=193 pw=0 time=50437 us)(object id 732786)
 190000      VIEW  VW_NSO_1 (cr=839 pr=399 pw=399 time=432108 us)
 190000       HASH JOIN  (cr=839 pr=399 pw=399 time=242105 us)
  50000        TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=34 us)
 200000        INDEX FAST FULL SCAN SALES_CUSTID_IDEX (cr=432 pr=0 pw=0 time=83 us)(object id 732783)
  70488     VIEW  VW_NSO_3 (cr=919 pr=0 pw=0 time=71295 us)
  70488      HASH JOIN  (cr=919 pr=0 pw=0 time=71290 us)
    500       TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=535 us)
 200000       TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=19 us)
```

Explanation:

	Baseline	Experiment #3	% Change	Performance Result
Count	1965	1962	-0.15%	better than the baseline
CPU	2.08	1.47	-29%	Better than the baseline
Elapsed	5.06	2.89	-43%	Better than the baseline
Disk	12530	6791	-46%	Better than the baseline
Query	10968	184766	1585%	Worse than the baseline
Current	3	14	367%	Worse than the baseline
Rows	29384	29384	0%	Same as the baseline

% Change Formula: (Index Experiment #3 - Baseline)/Baseline

The CPU time is 29% better than the baseline and elapsed time is 7% better than the baseline. The performance has improved a lot. However, the query opened an unreasonable number of blocks – 184766, about 6 times more than the rows returned, therefore, the performance is worse than experiment #1. This query read over 10 times the blocks more than the baseline. The parse statistics of CPU and elapsed time are both greater than 0.2, which are very high and unnecessary. From the trace file, we can see that the optimizer used some of the indexes. But it did not use all the indexes to execute the query.

Experiment #4:
To investigate further, I used hints to force the optimizer to use all the indexes I created.

Trace Result:

Trace Result

```
SELECT /*+ INDEX(c customer_pk customer_address_idx customer_fname_idx
    customer_sname_idx) INDEX(p product_pk product_pname_idx product_price_idx
    product_pdesc_idx) INDEX(s sales_pk sales_custid_idex sales_prodid_idex
    sales_qty_idx sales_sellprice_idx)*/
    c.custid,s.custid,p.prodid,c.first_name,c.surname,s.sellprice,c.address,s.qty,p.price
    FROM sales s, customer c, product p
    WHERE s.prodid=p.prodid AND s.custid=c.custid
    AND c.first_name = ANY (SELECT c.first_name
        FROM customer c WHERE c.first_name like 'C%' OR c.first_name like 'G%'
        OR c.first_name like 'F%' OR c.first_name like 'E%' OR c.first_name like 'J%')
    AND c.surname = ANY (SELECT c.surname FROM customer c
        WHERE c.surname like 'H%' OR c.surname like 'D%' OR c.surname like 'B%'
        OR c.surname like 'L%' OR c.surname like 'F%')
    AND s.sellprice > (SELECT min(s.sellprice) FROM sales s, product p
        WHERE s.prodid=p.prodid
        AND (p.pname like 'P%' OR p.pname like 'M%' OR p.pname like 'K%'))
    AND c.address = ANY (SELECT c.address
        FROM sales s, customer c WHERE s.custid= c.custid
        AND (c.address='AUCKLAND' OR c.address='WELLINGTON' OR c.address='NAPIER'))
    AND p.prodid = ANY (SELECT p.prodid FROM product p, sales s
        WHERE s.prodid= p.prodid AND p.price>2)
    AND s.qty = ANY (SELECT s.qty FROM sales s, product p
        WHERE s.prodid=p.prodid
        AND (p.pdesc like 'D2%' OR p.pdesc like 'D3%' OR p.pdesc like 'D5%'
        OR p.pdesc like 'D8%' OR p.pdesc like 'D7%'))
    AND p.pdesc = ANY (SELECT p.pdesc FROM product p
        WHERE p.pdesc like 'D2%' OR p.pdesc like 'D3%' OR p.pdesc like
        'D5%' OR p.pdesc like 'D8%' OR p.pdesc like 'D7%' OR p.pdesc like 'D9%' OR
        p.pdesc like 'D1%')
    AND p.price > (SELECT min(p.price) FROM sales s, product p
        WHERE s.prodid=p.prodid)
    ORDER BY p.prodid,s.saleid,c.custid, c.address
```

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.29       0.44         14         89          0           0
Execute      1      0.04       0.06        410        425          0           0
Fetch     1960      1.50       5.19       9515     184379         33       29384
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     1962      1.85       5.71       9939     184893         33       29384

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 10435

Rows     Row Source Operation
-------  ---------------------------------------------------
  29384  SORT ORDER BY (cr=184804 pr=9925 pw=5284 time=5234924 us)
  29384   HASH JOIN SEMI (cr=184804 pr=9373 pw=4732 time=4864100 us)
  29384    HASH JOIN SEMI (cr=183885 pr=8988 pw=4347 time=4689556 us)
  29384     HASH JOIN SEMI (cr=183046 pr=5361 pw=3255 time=3637019 us)
  29384      HASH JOIN SEMI (cr=182846 pr=4747 pw=2835 time=3247849 us)
 169362       HASH JOIN  (cr=182646 pr=2845 pw=1127 time=1768549 us)
 169362        HASH JOIN  (cr=182115 pr=1420 pw=0 time=700779 us)
    791         HASH JOIN SEMI (cr=1327 pr=415 pw=0 time=142192 us)
    990          HASH JOIN SEMI (cr=1320 pr=410 pw=0 time=122378 us)
    996           TABLE ACCESS BY INDEX ROWID PRODUCT (cr=895 pr=410 pw=0 time=66595 us)
    996            INDEX RANGE SCAN PRODUCT_PRICE_IDX (cr=428 pr=410 pw=0 time=63121 us)(object id 732791)
      1             SORT AGGREGATE (cr=425 pr=410 pw=0 time=63103 us)
 200000              HASH JOIN  (cr=425 pr=410 pw=0 time=800850 us)
   1000               TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=17 us)
 200000               INDEX FAST FULL SCAN SALES_PRODID_IDEX (cr=416 pr=410 pw=0 time=200137 us)(object id 732784)
 197303           VIEW  VW_NSO_2 (cr=425 pr=0 pw=0 time=197728 us)
 197303            HASH JOIN  (cr=425 pr=0 pw=0 time=424 us)
    990             TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=54 us)
 200000             INDEX FAST FULL SCAN SALES_PRODID_IDEX (cr=416 pr=0 pw=0 time=12 us)(object id 732784)
    800          INDEX FAST FULL SCAN PRODUCT_PDESC_IDX (cr=7 pr=5 pw=0 time=907 us)(object id 732792)
 199907         TABLE ACCESS BY INDEX ROWID SALES (cr=180788 pr=1005 pw=0 time=450419 us)
 199907          INDEX RANGE SCAN SALES_SELLPRICE_IDX (cr=1336 pr=1005 pw=0 time=250694 us)(object id 732789)
      1           SORT AGGREGATE (cr=919 pr=588 pw=0 time=50499 us)
  42234            HASH JOIN  (cr=919 pr=588 pw=0 time=42575 us)
    300             TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=329 us)
 200000             TABLE ACCESS FULL SALES (cr=910 pr=588 pw=0 time=41 us)
  60000        TABLE ACCESS BY INDEX ROWID CUSTOMER (cr=531 pr=298 pw=0 time=240096 us)
  60000         INDEX FULL SCAN CUSTOMER_PK (cr=125 pr=125 pw=0 time=60639 us)(object id 732780)
  30000       INDEX FAST FULL SCAN CUSTOMER_FNAME_IDX (cr=200 pr=194 pw=0 time=123178 us)(object id 732785)
  50000      INDEX FAST FULL SCAN CUSTOMER_SNAME_IDX (cr=200 pr=194 pw=0 time=50178 us)(object id 732786)
 190000     VIEW  VW_NSO_1 (cr=839 pr=1968 pw=420 time=853813 us)
 190000      HASH JOIN  (cr=839 pr=1968 pw=420 time=663809 us)
  50000       TABLE ACCESS FULL CUSTOMER (cr=407 pr=2 pw=0 time=43 us)
 200000       INDEX FAST FULL SCAN SALES_CUSTID_IDEX (cr=432 pr=426 pw=0 time=200149 us)(object id 732783)
  70488    VIEW  VW_NSO_3 (cr=919 pr=0 pw=0 time=141566 us)
  70488     HASH JOIN  (cr=919 pr=0 pw=0 time=71073 us)
    500      TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=525 us)
 200000      TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=10 us)
```

Explanation:

	Baseline	Experiment #4	% Change	Performance Result
Count	1965	1962	-0.15%	better than the baseline
CPU	2.08	1.85	-11%	Better than the baseline
Elapsed	5.06	5.71	13%	Worse than the baseline
Disk	12530	9939	-21%	Better than the baseline
Query	10968	184893	1586%	Worse than the baseline
Current	3	33	1000%	Worse than the baseline
Rows	29384	29384	0%	Same as the baseline

I found that when the optimizer used all of the indexes to execute the query, the performance got worse than only using part of the indexes. The number of buffers gotten for consistent read has increased vastly and is about 6 times of the rows returned and the number of buffers gotten in the current mode is 10 times more than the baseline. The reason that the indexes are not working well is that the query fetched a range of data. Data is stored in many different blocks. The server searches the index for each row and goes to different blocks to fetch this amount of data. This is very costly.

Conclusion:

Among all the experiments using indexes, experiment #1 with indexes only on primary keys and foreign keys and using only part of the indexes has the best performance. The overall performance of experiment #1 is better than the baseline. The CPU time has improved by 42% and the elapsed time has improved by 50%. The indexes can increase the performance when the selectivity of the predicate is high. Before creating indexes, we need to make sure the selectivity of the column is high enough to help with the performance because using indexes incorrectly may decrease the performance.

3. Indexed Clusters Setup

To improve the performance of the same query, I set up two indexed clusters.

Experiment #5:
The first experiment of cluster I set up is to cluster product table and sales table together.

SELECT count(*) FROM product; --1000 rows
SELECT count(*) FROM sales; --200000 rows
200000/1000 = 200
Therefore, 200 sales records in each product.
Size of product row: 10+6+15+8+5 = 44 bytes
Size of sales row: 10+8+5+10+8+10=51 bytes
Size of sales row * 200 sales records = 51 bytes * 200 = 10200 bytes
Therefore, Size of cluster=44+10200=10244 bytes

Creating Cluster Scripts:

```
--drop product and sales table and indexes created before on customer table
drop table product;
drop table sales;

drop index customer_address_idx;
drop index customer_fname_idx;
drop index customer_sname_idx;
alter table customer
drop constraint customer_pk;


CREATE CLUSTER sales_prod_cluster
    (prodid NUMBER(10))
    SIZE 10244;

CREATE TABLE product(
    ProdID  NUMBER(10) NOT NULL,
    PName   Varchar2(6),
    PDesc   Varchar2(15),
    Price   Number(8),
    QOH     Number(5))
    CLUSTER sales_prod_cluster(ProdID);

CREATE TABLE sales(
    SaleID  NUMBER(10) NOT NULL,
    SaleDate    DATE,
    Qty         Number(5),
    SellPrice   Number(10),
    CustID      NUMBER(8),
    ProdID      NUMBER(10))
    CLUSTER sales_prod_cluster(ProdID);

--create index on cluster
CREATE INDEX sales_prod_cluster_idx
    ON CLUSTER sales_prod_cluster;

--insert data from backup table to tables with clusters
INSERT into product select * from product_backup;
INSERT into sales select * from sales_backup;
```

Trace Result:

Trace Result

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.19       0.19          0         42          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     1960      0.99       2.38       6982      20030         49       29384
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     1962      1.18       2.57       6982      20072         49       29384

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 10435

Rows     Row Source Operation
-------  ---------------------------------------------------
  29384  SORT ORDER BY (cr=20030 pr=6982 pw=4532 time=2368442 us)
  29384   HASH JOIN SEMI (cr=20030 pr=6167 pw=3717 time=2072412 us)
  29384    HASH JOIN SEMI (cr=17739 pr=5754 pw=3304 time=1865920 us)
  29384     HASH JOIN SEMI (cr=16043 pr=3696 pw=2233 time=1237611 us)
  29384      HASH JOIN RIGHT SEMI (cr=15636 pr=3220 pw=1757 time=1085957 us)
  30000       TABLE ACCESS FULL CUSTOMER (cr=760 pr=0 pw=0 time=30118 us)
 169362       HASH JOIN  (cr=14876 pr=1127 pw=1127 time=804531 us)
 169362        NESTED LOOPS  (cr=14469 pr=0 pw=0 time=1591903 us)
    791         HASH JOIN SEMI (cr=10720 pr=0 pw=0 time=120725 us)
    796          HASH JOIN SEMI (cr=6384 pr=0 pw=0 time=55426 us)
    996           TABLE ACCESS FULL PRODUCT (cr=5095 pr=0 pw=0 time=53683 us)
      1            SORT AGGREGATE (cr=3806 pr=0 pw=0 time=51648 us)
 200000             NESTED LOOPS  (cr=3806 pr=0 pw=0 time=200049 us)
   1000              TABLE ACCESS FULL PRODUCT (cr=1289 pr=0 pw=0 time=6007 us)
 200000              TABLE ACCESS CLUSTER SALES (cr=2517 pr=0 pw=0 time=6493 us)
   1000               INDEX UNIQUE SCAN SALES_PROD_CLUSTER_IDX (cr=1002 pr=0 pw=0 time=3318 us)(object id 732829)
    800           TABLE ACCESS FULL PRODUCT (cr=1289 pr=0 pw=0 time=815 us)
 197303          VIEW  VW_NSO_2 (cr=4336 pr=0 pw=0 time=591959 us)
 197303           NESTED LOOPS  (cr=4336 pr=0 pw=0 time=394645 us)
    990            TABLE ACCESS FULL PRODUCT (cr=1289 pr=0 pw=0 time=2982 us)
 197303            TABLE ACCESS CLUSTER SALES (cr=3047 pr=0 pw=0 time=6932 us)
    990             INDEX UNIQUE SCAN SALES_PROD_CLUSTER_IDX (cr=992 pr=0 pw=0 time=3718 us)(object id 732829)
 169362         TABLE ACCESS CLUSTER SALES (cr=3749 pr=0 pw=0 time=15907 us)
    791          INDEX UNIQUE SCAN SALES_PROD_CLUSTER_IDX (cr=793 pr=0 pw=0 time=2474 us)(object id 732829)
      1          SORT AGGREGATE (cr=1891 pr=0 pw=0 time=11692 us)
  42234           NESTED LOOPS  (cr=1891 pr=0 pw=0 time=42346 us)
    300            TABLE ACCESS FULL PRODUCT (cr=1289 pr=0 pw=0 time=694 us)
  42234            TABLE ACCESS CLUSTER SALES (cr=602 pr=0 pw=0 time=1278 us)
    300             INDEX UNIQUE SCAN SALES_PROD_CLUSTER_IDX (cr=302 pr=0 pw=0 time=777 us)(object id 732829)
  60000       TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=60036 us)
  50000      TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=47 us)
 190000     VIEW  VW_NSO_1 (cr=1696 pr=399 pw=399 time=632313 us)
 190000      HASH JOIN  (cr=1696 pr=399 pw=399 time=442308 us)
  50000       TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=50064 us)
 200000       TABLE ACCESS FULL SALES (cr=1289 pr=0 pw=0 time=32 us)
  70488    VIEW  VW_NSO_3 (cr=2291 pr=0 pw=0 time=282021 us)
  70488     NESTED LOOPS  (cr=2291 pr=0 pw=0 time=141043 us)
    500      TABLE ACCESS FULL PRODUCT (cr=1289 pr=0 pw=0 time=1538 us)
  70488      TABLE ACCESS CLUSTER SALES (cr=1002 pr=0 pw=0 time=73770 us)
    500       INDEX UNIQUE SCAN SALES_PROD_CLUSTER_IDX (cr=502 pr=0 pw=0 time=2262 us)(object id 732829)
```

Explanation:

	Baseline	Experiment #5	% Change	Performance Result
Count	1965	1962	-0.15%	better than the baseline
CPU	2.08	1.18	-43%	Better than the baseline
Elapsed	5.06	2.57	-49%	Better than the baseline
Disk	12530	6982	-44%	Better than the baseline
Query	10968	20072	83%	Worse than the baseline
Current	3	49	1533%	Worse than the baseline
Rows	29384	29384	0%	Same as the baseline

After clustering the product and sales table, the performance has improved drastically. The CPU time is 43% better than the baseline. The elapsed time is about 49% better than the baseline. However, the parse statistics of CPU and elapsed time are still high and unnecessary, about 0.2 seconds. And the number of buffers gotten for consistent read has increased greatly and is now about the same size as the rows returned. It means that the optimizer read too many blocks to return the results.

Experiment #6:
To investigate further, I set up another cluster experiment. The second experiment of a cluster is to cluster sales table and customer table.

SELECT count(*) FROM customer; --60000 rows
SELECT count(*) FROM sales; --200000 rows
200000/60000 = 3
Therefore, 3 sales records in each customer.
Size of customer row: 8+15+15+20+12 = 70 bytes
Size of sales row: 10+8+5+10+8+10=51 bytes
Size of sales row * 3 sales records = 51 bytes * 3 = 153 bytes
Therefore, Size of cluster=70+153=223 bytes

Creating Cluster Scripts:

```
drop table product;
drop table sales;
drop table customer;
drop index sales_prod_cluster_idx;
drop cluster sales_prod_cluster;

--clustering sales and customer table
CREATE CLUSTER sales_cust_cluster
    (custid NUMBER(8))
    SIZE 223;

CREATE TABLE CUSTOMER (
    CustID      NUMBER(8) NOT NULL,
    FIRST_NAME  VARCHAR2(15),
    SURNAME     VARCHAR2(15),
    ADDRESS     VARCHAR2(20),
    PHONE_NUMBER NUMBER(12))
    CLUSTER sales_cust_cluster(CustID);

CREATE TABLE product(
    ProdID  NUMBER(10) NOT NULL,
    PName   Varchar2(6),
    PDesc   Varchar2(15),
    Price   Number(8),
    QOH     Number(5));

CREATE TABLE sales(
    SaleID  NUMBER(10) NOT NULL,
    SaleDate    DATE,
    Qty         Number(5),
    SellPrice   Number(10),
    CustID      NUMBER(8),
    ProdID      NUMBER(10))
    CLUSTER sales_cust_cluster(CustID);

--create index on cluster
CREATE INDEX sales_cust_cluster_idx
    ON CLUSTER sales_cust_cluster;

--insert data from backup table
INSERT into product select * from product_backup;
INSERT into sales select * from sales_backup;
INSERT into customer select * from customer_backup;

commit;
```

Trace Result:

Trace Result

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.16       0.29          0         42          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     1960      1.48       7.50       7127      80270        139       29384
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     1962      1.65       7.80       7127      80312        139       29384

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 10435

Rows     Row Source Operation
-------  ---------------------------------------------------
  29384  SORT ORDER BY (cr=80270 pr=7127 pw=4206 time=7484823 us)
  29384   HASH JOIN RIGHT SEMI (cr=80270 pr=5976 pw=3055 time=5846296 us)
    800    TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=65 us)
  29384   HASH JOIN SEMI (cr=80261 pr=5976 pw=3055 time=5728000 us)
  29384    HASH JOIN SEMI (cr=78006 pr=5507 pw=2586 time=5222519 us)
  29985     HASH JOIN SEMI (cr=75751 pr=5003 pw=2082 time=3889473 us)
  29985      HASH JOIN SEMI (cr=71259 pr=1405 pw=948 time=1826055 us)
  29985       HASH JOIN  (cr=69013 pr=901 pw=444 time=1156199 us)
    996        TABLE ACCESS FULL PRODUCT (cr=2264 pr=9 pw=0 time=123529 us)
      1        SORT AGGREGATE (cr=2255 pr=9 pw=0 time=123489 us)
 200000         HASH JOIN  (cr=2255 pr=9 pw=0 time=402485 us)
   1000          TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=25 us)
 200000          TABLE ACCESS FULL SALES (cr=2246 pr=9 pw=0 time=21 us)
  29985        NESTED LOOPS  (cr=66749 pr=892 pw=444 time=972287 us)
  30000         HASH JOIN  (cr=4492 pr=892 pw=444 time=618080 us)
  30000          SORT UNIQUE (cr=2246 pr=143 pw=143 time=265897 us)
  30000           TABLE ACCESS FULL CUSTOMER (cr=2246 pr=0 pw=0 time=40018 us)
  60000          TABLE ACCESS FULL CUSTOMER (cr=2246 pr=0 pw=0 time=60879 us)
  29985         TABLE ACCESS CLUSTER SALES (cr=62257 pr=0 pw=0 time=321297 us)
  30000          INDEX UNIQUE SCAN SALES_CUST_CLUSTER_IDX (cr=30002 pr=0 pw=0 time=110728 us)(object id 732837)
      1          SORT AGGREGATE (cr=2255 pr=0 pw=0 time=54303 us)
  42234           HASH JOIN  (cr=2255 pr=0 pw=0 time=42573 us)
    300            TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=334 us)
 200000            TABLE ACCESS FULL SALES (cr=2246 pr=0 pw=0 time=9 us)
  50000        TABLE ACCESS FULL CUSTOMER (cr=2246 pr=0 pw=0 time=51460 us)
 190000       VIEW  VW_NSO_1 (cr=4492 pr=1540 pw=420 time=1309153 us)
 190000        HASH JOIN  (cr=4492 pr=1540 pw=420 time=929139 us)
  50000         TABLE ACCESS FULL CUSTOMER (cr=2246 pr=0 pw=0 time=50858 us)
 200000         TABLE ACCESS FULL SALES (cr=2246 pr=0 pw=0 time=200070 us)
 197303       VIEW  VW_NSO_2 (cr=2255 pr=0 pw=0 time=395499 us)
 197303        HASH JOIN  (cr=2255 pr=0 pw=0 time=198191 us)
    990         TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=1078 us)
 200000         TABLE ACCESS FULL SALES (cr=2246 pr=0 pw=0 time=25 us)
  70488      VIEW  VW_NSO_3 (cr=2255 pr=0 pw=0 time=142036 us)
  70488       HASH JOIN  (cr=2255 pr=0 pw=0 time=71542 us)
    500        TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=1048 us)
 200000        TABLE ACCESS FULL SALES (cr=2246 pr=0 pw=0 time=25 us)
```

Explanation:

	Baseline	Experiment #6	% Change	Performance Result
Count	1965	1962	-0.15%	better than the baseline
CPU	2.08	1.65	-21%	better than the baseline
Elapsed	5.06	7.80	54%	Worse than the baseline
Disk	12530	7127	-71%	better than the baseline
Query	10968	80312	8455%	Worse than the baseline
Current	3	139	80%	Worse than the baseline
Rows	29384	29384	0%	Same as the baseline

As we can see from the trace file, the optimizer accessed the cluster customer when needed and performed a full table scan on other operations. The query is about 4 times the rows returned. The optimizer has read too many blocks to return the rows. The current has increased by 80%. The CPU time is decreased by 21% but the elapsed time is increased by 54%. The overall performance is worse than the baseline.

Conclusion:

Because this is a complex query. It joins three tables to fetch some data and also joins two tables, product and sales, sales and customer to fetch some rows.
In experiment #5, we cluster product and sales tables. But the query joins three tables together and in some subqueries, it joins sales and customer tables together. The cluster only works well when we only join sales and product together.
In experiment #6, we cluster customer and sales tables. But the query joins three tables together and in some subqueries, it joins sales and product tables together. In addition, in some subqueries, the query fetches a large amount of data from only one table. The optimizer is performing a full table scan for the step fetching a large amount of data from one table, which decreases the performance and has a much higher cost when the table is in a cluster and needs a full table scan. This query has many non-equality operators, which affect the performance of the cluster.

To conclude, when using the cluster, we need to make sure if we use the cluster, we need to use the correct cluster type and the query mostly includes join tables and does not perform a full table scan.

4. Hash Clusters Setup

To improve the performance of the same query, I set up two hash clusters.

Experiment #7:
The first experiment of hash cluster I set up is to cluster product table and sales table together and I used the hint to force the optimizer to use the cluster just set up.

Because there are 1000 products. Therefore hashkeys = 1000;

Creating Cluster Scripts:

```
/*clustering sales and product table
Currently we have 1000 product. assume that the number of product in the
future will be 1000*/
CREATE CLUSTER sales_prod_hashCluster
    (prodid NUMBER(10))
    SIZE 10244 HASHKEYS 1000;

CREATE TABLE CUSTOMER (
    CustID      NUMBER(8) NOT NULL,
    FIRST_NAME  VARCHAR2(15),
    SURNAME     VARCHAR2(15),
    ADDRESS     VARCHAR2(20),
    PHONE_NUMBER NUMBER(12));

CREATE TABLE product(
    ProdID  NUMBER(10) NOT NULL,
    PName   Varchar2(6),
    PDesc   Varchar2(15),
    Price   Number(8),
    QOH     Number(5))
    CLUSTER sales_prod_hashCluster(prodid);

CREATE TABLE sales(
    SaleID  NUMBER(10) NOT NULL,
    SaleDate    DATE,
    Qty         Number(5),
    SellPrice   Number(10),
    CustID      NUMBER(8),
    ProdID      NUMBER(10))
    CLUSTER sales_prod_hashCluster(prodid);


--insert data from backup table
INSERT into product select * from product_backup;
INSERT into sales select * from sales_backup;
INSERT into customer select * from customer_backup;

commit;
```

Trace Rsult:

Trace Result

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.24       0.37          0         52          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     1960      1.34       6.23       8749      24331         63       29384
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     1962      1.58       6.60       8749      24383         63       29384

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 10435

Rows     Row Source Operation
-------  ---------------------------------------------------
  29384  SORT ORDER BY (cr=24331 pr=8749 pw=4669 time=6206583 us)
  29384   HASH JOIN SEMI (cr=24331 pr=7818 pw=3738 time=5135112 us)
  29384    HASH JOIN SEMI (cr=21445 pr=7405 pw=3325 time=4639949 us)
  29384     HASH JOIN SEMI (cr=19310 pr=4206 pw=2233 time=3165341 us)
  29384      HASH JOIN RIGHT SEMI (cr=18903 pr=3730 pw=1757 time=2540800 us)
  30000       TABLE ACCESS FULL CUSTOMER (cr=746 pr=0 pw=0 time=49 us)
 169362       HASH JOIN  (cr=18157 pr=1637 pw=1127 time=1784264 us)
 169362        NESTED LOOPS  (cr=17750 pr=510 pw=0 time=477086 us)
    791         HASH JOIN SEMI (cr=13174 pr=510 pw=0 time=244821 us)
    796          HASH JOIN SEMI (cr=7989 pr=510 pw=0 time=108910 us)
    996           TABLE ACCESS FULL PRODUCT (cr=6261 pr=510 pw=0 time=103768 us)
      1            SORT AGGREGATE (cr=4533 pr=510 pw=0 time=102711 us)
 200000             NESTED LOOPS  (cr=4533 pr=510 pw=0 time=200038 us)
   1000              TABLE ACCESS FULL PRODUCT (cr=1728 pr=510 pw=0 time=6011 us)
 200000              TABLE ACCESS HASH SALES (cr=2805 pr=0 pw=0 time=2617 us)
    800           TABLE ACCESS FULL PRODUCT (cr=1728 pr=0 pw=0 time=1613 us)
 197303          VIEW  VW_NSO_2 (cr=5185 pr=0 pw=0 time=1578465 us)
 197303           NESTED LOOPS  (cr=5185 pr=0 pw=0 time=789249 us)
    990            TABLE ACCESS FULL PRODUCT (cr=1728 pr=0 pw=0 time=6946 us)
 197303            TABLE ACCESS HASH SALES (cr=3457 pr=0 pw=0 time=396084 us)
 169362         TABLE ACCESS HASH SALES (cr=4576 pr=0 pw=0 time=197027 us)
      1          SORT AGGREGATE (cr=2395 pr=0 pw=0 time=26248 us)
  42234           NESTED LOOPS  (cr=2395 pr=0 pw=0 time=42254 us)
    300            TABLE ACCESS FULL PRODUCT (cr=1728 pr=0 pw=0 time=2406 us)
  42234            TABLE ACCESS HASH SALES (cr=667 pr=0 pw=0 time=690 us)
  60000        TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=60067 us)
  50000      TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=40 us)
190000     VIEW  VW_NSO_1 (cr=2135 pr=1540 pw=420 time=1131405 us)
190000      HASH JOIN  (cr=2135 pr=1540 pw=420 time=941403 us)
  50000       TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=50041 us)
200000       TABLE ACCESS FULL SALES (cr=1728 pr=0 pw=0 time=200041 us)
  70488    VIEW  VW_NSO_3 (cr=2886 pr=0 pw=0 time=282019 us)
  70488     NESTED LOOPS  (cr=2886 pr=0 pw=0 time=211526 us)
    500      TABLE ACCESS FULL PRODUCT (cr=1728 pr=0 pw=0 time=3039 us)
  70488      TABLE ACCESS HASH SALES (cr=1158 pr=0 pw=0 time=71640 us)
```

Explanation:

	Baseline	Experiment #7	% Change	Performance Result
Count	1965	1962	-0.15%	Better than the Baseline
CPU	2.08	1.58	-24%	Better than the baseline
Elapsed	5.06	6.60	30%	Worse than the baseline
Disk	12530	8749	-30%	Better than the baseline
Query	10968	24383	122%	Worse than the baseline
Current	3	63	2000%	Worse than the baseline
Rows	29384	29384	0%	Same as the baseline

As we can see from the trace file, the CPU time has 28% improvement while the elapsed time has increased by about 30%. In this query, product and sales are joined together. The number of buffers gotten in the current mode is 2 times of the baseline. From the trace file, we find out the optimizer has to perform a full table scan on some operations, which makes the cluster not working well.

Experiment #8:

In this experiment, I cluster customer and sales tables with hash key custid and used hints to force the optimizer using the cluster created.
Because there are 60000 customers. Therefore hashkeys = 60000;

Creating Hash Cluster Scripts:

```
/*clustering sales and customer table
Currently we have 60000 customers. assume that the number of customers in the
future will be 60000*/
CREATE CLUSTER sales_cust_hashCluster
    (custid NUMBER(8))
    SIZE 223 HASHKEYS 60000;

CREATE TABLE CUSTOMER (
    CustID      NUMBER(8) NOT NULL,
    FIRST_NAME  VARCHAR2(15),
    SURNAME     VARCHAR2(15),
    ADDRESS     VARCHAR2(20),
    PHONE_NUMBER NUMBER(12))
    CLUSTER sales_cust_hashCluster(CustID);

CREATE TABLE product(
    ProdID  NUMBER(10) NOT NULL,
    PName   Varchar2(6),
    PDesc   Varchar2(15),
    Price   Number(8),
    QOH     Number(5));

CREATE TABLE sales(
    SaleID  NUMBER(10) NOT NULL,
    SaleDate    DATE,
    Qty         Number(5),
    SellPrice   Number(10),
    CustID      NUMBER(8),
    ProdID      NUMBER(10))
    CLUSTER sales_cust_hashCluster(CustID);


--insert data from backup table
INSERT into product select * from product_backup;
INSERT into sales select * from sales_backup;
INSERT into customer select * from customer_backup;

commit;
```

Trace Result:

Trace Result

```
SELECT /*+ cluster(sales_cust_hashCluster) */ c.custid,s.custid,p.prodid,
    c.first_name,c.surname,s.sellprice,c.address,s.qty,p.price
    FROM sales s, customer c, product p
    WHERE s.prodid=p.prodid AND s.custid=c.custid
    AND c.first_name = ANY (SELECT c.first_name FROM customer c
        WHERE c.first_name like 'C%' OR c.first_name like 'G%' OR c.first_name like 'F%'
        OR c.first_name like 'E%' OR c.first_name like 'J%')
    AND c.surname = ANY (SELECT c.surname FROM customer c WHERE c.surname like 'H%' OR c.surname
        like 'D%' OR c.surname like 'B%' OR c.surname like 'L%' OR c.surname like 'F%')
    AND s.sellprice > (SELECT min(s.sellprice) FROM sales s, product p
        WHERE s.prodid=p.prodid
        AND (p.pname like 'P%' OR p.pname like 'M%' OR p.pname like 'K%'))
    AND c.address = ANY (SELECT c.address FROM sales s, customer c WHERE s.custid= c.custid
        AND (c.address='AUCKLAND' OR c.address='WELLINGTON' OR c.address='NAPIER'))
    AND p.prodid = ANY (SELECT p.prodid FROM product p, sales s
        WHERE s.prodid= p.prodid AND p.price>2)
    AND s.qty = ANY (SELECT s.qty FROM sales s, product p
        WHERE s.prodid=p.prodid AND (p.pdesc like 'D2%' OR p.pdesc like 'D3%'
        OR p.pdesc like 'D5%' OR p.pdesc like 'D8%' OR p.pdesc like 'D7%'))
    AND p.pdesc = ANY (SELECT p.pdesc FROM product p
        WHERE p.pdesc like 'D2%' OR p.pdesc like 'D3%' OR p.pdesc like 'D5%'
        OR p.pdesc like 'D8%' OR p.pdesc like 'D7%' OR p.pdesc like 'D9%' OR
        p.pdesc like 'D1%')
    AND p.price > (SELECT min(p.price) FROM sales s, product p
        WHERE s.prodid=p.prodid)
    ORDER BY p.prodid,s.saleid,c.custid, c.address

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.17       0.28          0         46          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     1960      2.15       6.34       4575     249390         23       29384
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     1962      2.33       6.63       4575     249436         23       29384

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 10435

Rows     Row Source Operation
-------  ---------------------------------------------------
  29384  SORT ORDER BY (cr=249390 pr=4575 pw=3588 time=6320567 us)
  29384   HASH JOIN SEMI (cr=249390 pr=4172 pw=3185 time=5859191 us)
  29384    HASH JOIN SEMI (cr=247657 pr=3752 pw=2765 time=5288035 us)
  29384     HASH JOIN SEMI (cr=192573 pr=2093 pw=2093 time=3941026 us)
  29384      HASH JOIN SEMI (cr=190831 pr=1708 pw=1708 time=3572427 us)
 169362       NESTED LOOPS  (cr=189098 pr=0 pw=0 time=1960112 us)
 169362        HASH JOIN  (cr=6977 pr=0 pw=0 time=435848 us)
    791         HASH JOIN SEMI (cr=3502 pr=0 pw=0 time=204630 us)
    990          HASH JOIN SEMI (cr=3493 pr=0 pw=0 time=99177 us)
    996           TABLE ACCESS FULL PRODUCT (cr=1751 pr=0 pw=0 time=93225 us)
      1            SORT AGGREGATE (cr=1742 pr=0 pw=0 time=93179 us)
 200000             HASH JOIN  (cr=1742 pr=0 pw=0 time=800445 us)
   1000              TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=25 us)
 200000              TABLE ACCESS FULL SALES (cr=1733 pr=0 pw=0 time=200012 us)
 197303           VIEW  VW_NSO_2 (cr=1742 pr=0 pw=0 time=592442 us)
 197303            HASH JOIN  (cr=1742 pr=0 pw=0 time=533 us)
    990             TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=22 us)
 200000             TABLE ACCESS FULL SALES (cr=1733 pr=0 pw=0 time=400009 us)
    800          TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=822 us)
 199907         TABLE ACCESS FULL SALES (cr=3475 pr=0 pw=0 time=262753 us)
      1          SORT AGGREGATE (cr=1742 pr=0 pw=0 time=61807 us)
  42234           HASH JOIN  (cr=1742 pr=0 pw=0 time=380476 us)
    300            TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=350 us)
 200000            TABLE ACCESS FULL SALES (cr=1733 pr=0 pw=0 time=7 us)
 169362        TABLE ACCESS HASH CUSTOMER (cr=182121 pr=0 pw=0 time=798681 us)
  30000       TABLE ACCESS FULL CUSTOMER (cr=1733 pr=0 pw=0 time=30057 us)
  70488      VIEW  VW_NSO_3 (cr=1742 pr=0 pw=0 time=141714 us)
  70488       HASH JOIN  (cr=1742 pr=0 pw=0 time=71224 us)
    500        TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=542 us)
 200000        TABLE ACCESS FULL SALES (cr=1733 pr=0 pw=0 time=23 us)
 190000     VIEW  VW_NSO_1 (cr=55084 pr=0 pw=0 time=1140063 us)
 190000      NESTED LOOPS  (cr=55084 pr=0 pw=0 time=950060 us)
  50000       TABLE ACCESS FULL CUSTOMER (cr=1733 pr=0 pw=0 time=200044 us)
 190000       TABLE ACCESS HASH SALES (cr=53351 pr=0 pw=0 time=374207 us)
  50000    TABLE ACCESS FULL CUSTOMER (cr=1733 pr=0 pw=0 time=250044 us)
```

Explanation:

	Baseline	Experiment #8	% Change	Performance Result
Count	1965	1962	-0.15%	Better than the baseline
CPU	2.08	2.33	12%	Worse than the baseline
Elapsed	5.06	6.63	31%	Worse than the baseline
Disk	12530	4575	-63%	Better than the baseline
Query	10968	249436	2174%	Worse than the baseline
Current	3	23	667%	Worse than the baseline
Rows	29384	29384	0%	Same as the baseline

From the trace file, we found that the number of blocks the query read has increased a lot and now is about 10 times of the rows returned. This experiment is reading too many blocks to fetch the data. And the CPU time increased by 12% and elapsed time increased by over 30%. We can see from the trace file that the optimizer fetched from the hash cluster and also did a full table scan in some operations. Because the hash cluster does not support a full table scan, the performance got worse.

Conclusions:

The performance of hash clustering product and sales table has better performance than the one clustering sales and customer table. However, the performance of both of them is worse than the baseline. Hash cluster has a better performance when the hash key column is queried frequently with equality conditions. As we have column prodid in the WHERE clause, the hash cluster on the product and sales table has better performance than the one clustering sales and customer table. However, the overall performance of these two clusters is not better than the baseline. The reason is that in our query, we only have one column - prodid in the WHERE clause and all other predicates are not hash keys. Furthermore, the query needs range searching, which is not supported in hashing. To conclude, if we have other conditions that require a full table scan, a hash cluster is not a good option.

5. Statement Tuning

After conducting all these experiments, we found that experiment #1 where we set up indexed for primary key and foreign keys on product, sales and customer has the best performance. For further improvement, we will use experiment #1 and improve it by inserting hints and swapping the table orders.

Experiment #9: Table Order

First of all, I will swap the order of the tables in the 'from' clause of the query to change the driving table to improve the performance. The total number of rows in the product table is 1000. The total number of rows in the customer table is 60000. The total number of rows in the sales table is 200000. The size of the product table is smaller than that of the customer table. And the size of the customer table is smaller than that of the sales table. In this experiment, I swap the orders of the tables to make the product the driver for the subquery that joins only product and sales. And then I will swap the orders of the tables in the subquery to make the customer the driver when the subquery joins only customer and sales. For the main three table joins, I will use the pair (product and sales) that gives the smallest number of rows and then join it with the customer table.

Trace Result:

Trace Result

```
/*swap the table orders to make the table with smaller number of rows to be
    the driving table*/
SELECT c.custid,s.custid,p.prodid,c.first_name,c.surname,s.sellprice,
    c.address,s.qty,p.price
    FROM product p, sales s, customer c
    WHERE s.prodid=p.prodid AND s.custid=c.custid
    AND c.first_name = ANY (SELECT c.first_name FROM customer c
        WHERE c.first_name like 'C%' OR c.first_name like 'G%' OR c.first_name like 'F%'
        OR c.first_name like 'E%' OR c.first_name like 'J%')
    AND c.surname = ANY (SELECT c.surname FROM customer c
        WHERE c.surname like 'H%' OR c.surname like 'D%' OR c.surname like 'B%'
        OR c.surname like 'L%' OR c.surname like 'F%')
    AND s.sellprice > (SELECT min(s.sellprice) FROM product p, sales s
        WHERE p.prodid= s.prodid
        AND (p.pname like 'P%' OR p.pname like 'M%' OR p.pname like 'K%'))
    AND c.address = ANY (SELECT c.address FROM customer c, sales s
        WHERE c.custid= s.custid
        AND (c.address='AUCKLAND' OR c.address='WELLINGTON' OR c.address='NAPIER'))
    AND p.prodid = ANY (SELECT p.prodid FROM product p, sales s
        WHERE p.prodid= s.prodid AND p.price>2)
    AND s.qty = ANY (SELECT s.qty FROM product p, sales s
        WHERE p.prodid= s.prodid
        AND (p.pdesc like 'D2%' OR p.pdesc like 'D3%' OR p.pdesc like 'D5%'
        OR p.pdesc like 'D8%' OR p.pdesc like 'D7%'))
    AND p.pdesc = ANY (SELECT p.pdesc FROM product p
        WHERE p.pdesc like 'D2%' OR p.pdesc like 'D3%' OR p.pdesc like
        'D5%' OR p.pdesc like 'D8%' OR p.pdesc like 'D7%' OR p.pdesc like 'D9%' OR
        p.pdesc like 'D1%')
    AND p.price > (SELECT min(p.price) FROM product p, sales s
        WHERE p.prodid= s.prodid)
    ORDER BY p.prodid,s.saleid,c.custid, c.address
```

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.16       0.16          0         75          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     1960      0.97       2.05       5767       5676          8       29384
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     1962      1.14       2.22       5767       5751          8       29384

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 10435

Rows     Row Source Operation
-------  ---------------------------------------------------
  29384  SORT ORDER BY (cr=5676 pr=5767 pw=5116 time=2045924 us)
  29384   HASH JOIN SEMI (cr=5676 pr=5483 pw=4832 time=1855142 us)
  29384    HASH JOIN SEMI (cr=4757 pr=5033 pw=4382 time=1765134 us)
  29384     HASH JOIN SEMI (cr=4350 pr=4557 pw=3906 time=1629957 us)
  29384      HASH JOIN SEMI (cr=3511 pr=2835 pw=2835 time=1171089 us)
 169362       HASH JOIN  (cr=3104 pr=1127 pw=1127 time=733922 us)
 169362        HASH JOIN  (cr=2697 pr=0 pw=0 time=361621 us)
    791         HASH JOIN SEMI (cr=868 pr=0 pw=0 time=145443 us)
    990          HASH JOIN SEMI (cr=859 pr=0 pw=0 time=135591 us)
    996           TABLE ACCESS FULL PRODUCT (cr=434 pr=0 pw=0 time=75106 us)
      1            SORT AGGREGATE (cr=425 pr=0 pw=0 time=75064 us)
 200000             HASH JOIN  (cr=425 pr=0 pw=0 time=600518 us)
   1000              TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=22 us)
 200000              INDEX FAST FULL SCAN SALES_PRODID_IDEX (cr=416 pr=0 pw=0 time=200046 us)(object id 732957)
 197303           VIEW  VW_NSO_2 (cr=425 pr=0 pw=0 time=197714 us)
 197303            HASH JOIN  (cr=425 pr=0 pw=0 time=405 us)
    990             TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=26 us)
 200000             INDEX FAST FULL SCAN SALES_PRODID_IDEX (cr=416 pr=0 pw=0 time=11 us)(object id 732957)
    800          TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=18 us)
 199907         TABLE ACCESS FULL SALES (cr=1829 pr=0 pw=0 time=46830 us)
      1          SORT AGGREGATE (cr=919 pr=0 pw=0 time=46809 us)
  42234           HASH JOIN  (cr=919 pr=0 pw=0 time=42552 us)
    300            TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=22 us)
 200000            TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=6 us)
  60000        TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=60057 us)
  30000       TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=30043 us)
 190000      VIEW  VW_NSO_1 (cr=839 pr=399 pw=399 time=446423 us)
 190000       HASH JOIN  (cr=839 pr=399 pw=399 time=256419 us)
  50000        TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=50106 us)
 200000        INDEX FAST FULL SCAN SALES_CUSTID_IDEX (cr=432 pr=0 pw=0 time=39 us)(object id 732956)
  50000     TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=31 us)
  70488    VIEW  VW_NSO_3 (cr=919 pr=0 pw=0 time=141755 us)
  70488     HASH JOIN  (cr=919 pr=0 pw=0 time=71265 us)
    500      TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=529 us)
 200000      TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=16 us)
```

Explanation:

	Baseline	Experiment #9	% Change	Performance Result
Count	1965	1962	-0.15%	Better than the baseline
CPU	2.08	1.14	-45%	Better than the baseline
Elapsed	5.06	2.22	-56%	Better than the baseline
Disk	12530	5767	-54%	Better than the baseline
Query	10968	5751	-48%	Better than the baseline
Current	3	8	167%	Worse than the baseline
Rows	29384	29384	0%	Same as the baseline

	Experiment #1	Experiment #9	% Change	Performance Result
Count	1962	1962	0%	Same as the Experiment #1
CPU	1.2	1.14	-5%	Better than the Experiment #1
Elapsed	2.51	2.22	-12%	Better than the Experiment #1
Disk	6552	5767	-12%	Better than the Experiment #1
Query	5753	5751	0%	Same as the Experiment #1
Current	8	8	0%	Same as the Experiment #1
Rows	29384	29384	0%	Same as the Experiment #1

From the trace file, we found that the CPU time is 45% better than the baseline while the elapsed time is 56% better than the baseline. Compared with Experiment #1, after swapping table orders, the performance has been improved slightly. The CPU time is 5% better than experiment #1. The elapsed time is 12% better than experiment #1.

Conclusions:

To tune the performance, we should select the most efficient driving table. In this case, for the subquery that joins the product and sales table, the product should be used as the driver. For the subquery that joins the customer and sales table, the customer should be used as the driver.

Experiment #10: Using Hint 1

I used the table orders adjusted from Experiment #9 and continued tuning the query by using hints. I have done many experiments by using a hint to force optimizer using a single index and found that when using a hint to force the optimizer using the index product_pk has the best performance.

Trace Result

```
SELECT /*+ INDEX(p product_pk) */ c.custid,s.custid,p.prodid,c.first_name,
    c.surname,s.sellprice,c.address,s.qty,p.price
    FROM product p, sales s, customer c
    WHERE s.prodid=p.prodid AND s.custid=c.custid
    AND c.first_name = ANY (SELECT c.first_name FROM customer c
        WHERE c.first_name like 'C%' OR c.first_name like 'G%' OR c.first_name like 'F%'
        OR c.first_name like 'E%' OR c.first_name like 'J%')
    AND c.surname = ANY (SELECT c.surname FROM customer c
        WHERE c.surname like 'H%' OR c.surname like 'D%' OR c.surname like 'B%'
        OR c.surname like 'L%' OR c.surname like 'F%')
    AND s.sellprice > (SELECT min(s.sellprice) FROM product p, sales s
        WHERE p.prodid= s.prodid
        AND (p.pname like 'P%' OR p.pname like 'M%' OR p.pname like 'K%'))
    AND c.address = ANY (SELECT c.address
        FROM customer c, sales s WHERE c.custid= s.custid
        AND (c.address='AUCKLAND' OR c.address='WELLINGTON' OR c.address='NAPIER'))
    AND p.prodid = ANY (SELECT p.prodid FROM product p, sales s
        WHERE p.prodid= s.prodid AND p.price>2)
    AND s.qty = ANY (SELECT s.qty FROM product p, sales s
        WHERE p.prodid= s.prodid
        AND (p.pdesc like 'D2%' OR p.pdesc like 'D3%' OR p.pdesc like 'D5%'
        OR p.pdesc like 'D8%' OR p.pdesc like 'D7%'))
    AND p.pdesc = ANY (SELECT p.pdesc FROM product p
        WHERE p.pdesc like 'D2%' OR p.pdesc like 'D3%' OR p.pdesc like 'D5%'
        OR p.pdesc like 'D8%' OR p.pdesc like 'D7%' OR p.pdesc like 'D9%' OR
        p.pdesc like 'D1%')
    AND p.price > (SELECT min(p.price) FROM product p, sales s
        WHERE p.prodid= s.prodid)
    ORDER BY p.prodid,s.saleid,c.custid, c.address


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     1960      0.91       2.03       6066       5683          9       29384
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     1962      0.91       2.03       6066       5683          9       29384

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 10435

Rows     Row Source Operation
-------  ---------------------------------------------------
  29384  SORT ORDER BY (cr=5683 pr=6066 pw=5079 time=2027079 us)
  29384   HASH JOIN SEMI (cr=5683 pr=5782 pw=4795 time=1794064 us)
  29384    HASH JOIN SEMI (cr=4764 pr=5397 pw=4410 time=1707251 us)
  29384     HASH JOIN SEMI (cr=4357 pr=4921 pw=3934 time=1542813 us)
  29384      HASH JOIN SEMI (cr=3518 pr=2863 pw=2863 time=1219372 us)
 169362       HASH JOIN  (cr=3111 pr=1127 pw=1127 time=793620 us)
 169362        HASH JOIN  (cr=2704 pr=0 pw=0 time=338620 us)
    791         HASH JOIN SEMI (cr=875 pr=0 pw=0 time=131904 us)
    990          HASH JOIN SEMI (cr=866 pr=0 pw=0 time=120436 us)
    996           TABLE ACCESS BY INDEX ROWID PRODUCT (cr=441 pr=0 pw=0 time=58987 us)
   1000            INDEX FULL SCAN PRODUCT_PK (cr=3 pr=0 pw=0 time=59 us)(object id 732954)
      1            SORT AGGREGATE (cr=425 pr=0 pw=0 time=57900 us)
 200000             HASH JOIN  (cr=425 pr=0 pw=0 time=600728 us)
   1000              TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=1033 us)
 200000              INDEX FAST FULL SCAN SALES_PRODID_IDEX (cr=416 pr=0 pw=0 time=26 us)(object id 732957)
 197303           VIEW  VW_NSO_2 (cr=425 pr=0 pw=0 time=197663 us)
 197303            HASH JOIN  (cr=425 pr=0 pw=0 time=357 us)
    990             TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=14 us)
 200000             INDEX FAST FULL SCAN SALES_PRODID_IDEX (cr=416 pr=0 pw=0 time=14 us)(object id 732957)
    800         TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=817 us)
 199907        TABLE ACCESS FULL SALES (cr=1829 pr=0 pw=0 time=37419 us)
      1         SORT AGGREGATE (cr=919 pr=0 pw=0 time=37401 us)
  42234          HASH JOIN  (cr=919 pr=0 pw=0 time=42570 us)
    300           TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=28 us)
 200000           TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=10 us)
  60000       TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=36 us)
  30000      TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=31 us)
 190000     VIEW  VW_NSO_1 (cr=839 pr=399 pw=399 time=232609 us)
 190000      HASH JOIN  (cr=839 pr=399 pw=399 time=42607 us)
  50000       TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=22 us)
 200000       INDEX FAST FULL SCAN SALES_CUSTID_IDEX (cr=432 pr=0 pw=0 time=20 us)(object id 732956)
  50000     TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=28 us)
  70488    VIEW  VW_NSO_3 (cr=919 pr=0 pw=0 time=282768 us)
  70488     HASH JOIN  (cr=919 pr=0 pw=0 time=71303 us)
    500      TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=530 us)
 200000      TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=18 us)
```

Explanation:

	Baseline	Experiment #10	% Change	Performance Result
Count	1965	1962	-0.15%	Better than the baseline
CPU	2.08	0.91	-56%	Better than the baseline
Elapsed	5.06	2.03	-60%	Better than the baseline
Disk	12530	6066	-52%	Better than the baseline
Query	10968	5683	-48%	Better than the baseline
Current	3	9	200%	Worse than the baseline
Rows	29384	29384	0%	Same as the baseline

We found that by using this hint, the CPU time is 56% better than the baseline and the elapsed time is 60% better than the baseline. The disk and query are improved a lot as well. The query opened only 5683 blocks to get the results. However, the current has increased and is now 3 times more than the baseline.

Experiment #11: Using Hint 2

In this experiment, I used the hint - PARALLEL_INDEX to allow parallelization of a fast full index scan on any index.

Trace Result

```
SELECT /*+ PARALLEL_INDEX */ c.custid,s.custid,p.prodid,c.first_name,
    c.surname,s.sellprice,c.address,s.qty,p.price
    FROM product p, sales s, customer c
    WHERE s.prodid=p.prodid AND s.custid=c.custid
    AND c.first_name = ANY (SELECT c.first_name FROM customer c
        WHERE c.first_name like 'C%' OR c.first_name like 'G%' OR c.first_name like 'F%'
        OR c.first_name like 'E%' OR c.first_name like 'J%')
    AND c.surname = ANY (SELECT c.surname FROM customer c
        WHERE c.surname like 'H%' OR c.surname like 'D%' OR c.surname like 'B%'
        OR c.surname like 'L%' OR c.surname like 'F%')
    AND s.sellprice > (SELECT min(s.sellprice) FROM product p, sales s
        WHERE p.prodid= s.prodid
        AND (p.pname like 'P%' OR p.pname like 'M%' OR p.pname like 'K%'))
    AND c.address = ANY (SELECT c.address FROM customer c, sales s
        WHERE c.custid= s.custid AND (c.address='AUCKLAND' OR c.address=
        'WELLINGTON' OR c.address='NAPIER'))
    AND p.prodid = ANY (SELECT p.prodid FROM product p, sales s
        WHERE p.prodid= s.prodid AND p.price>2)
    AND s.qty = ANY (SELECT s.qty FROM product p, sales s
        WHERE p.prodid= s.prodid
        AND (p.pdesc like 'D2%' OR p.pdesc like 'D3%' OR p.pdesc like 'D5%'
        OR p.pdesc like 'D8%' OR p.pdesc like 'D7%'))
    AND p.pdesc = ANY (SELECT p.pdesc FROM product p
        WHERE p.pdesc like 'D2%' OR p.pdesc like 'D3%' OR p.pdesc like
        'D5%' OR p.pdesc like 'D8%' OR p.pdesc like 'D7%' OR p.pdesc like 'D9%'
        OR p.pdesc like 'D1%')
    AND p.price > (SELECT min(p.price) FROM product p, sales s
        WHERE p.prodid= s.prodid)
    ORDER BY p.prodid,s.saleid,c.custid, c.address
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	1960	0.93	1.84	7332	5676	8	29384
total	1962	0.93	1.84	7332	5676	8	29384

```
Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 10435

Rows     Row Source Operation
-------  ---------------------------------------------------
  29384  SORT ORDER BY (cr=5676 pr=7332 pw=5225 time=1829937 us)
  29384   HASH JOIN SEMI (cr=5676 pr=7048 pw=4941 time=1642851 us)
  29384    HASH JOIN SEMI (cr=4757 pr=6643 pw=4536 time=1564473 us)
  29384     HASH JOIN SEMI (cr=4350 pr=6167 pw=4060 time=1409722 us)
  29384      HASH JOIN SEMI (cr=3511 pr=2968 pw=2968 time=1078175 us)
 169362       HASH JOIN  (cr=3104 pr=1232 pw=1232 time=707327 us)
 169362        HASH JOIN  (cr=2697 pr=0 pw=0 time=342137 us)
    791         HASH JOIN SEMI (cr=868 pr=0 pw=0 time=130692 us)
    990          HASH JOIN SEMI (cr=859 pr=0 pw=0 time=120300 us)
    996           TABLE ACCESS FULL PRODUCT (cr=434 pr=0 pw=0 time=59080 us)
      1            SORT AGGREGATE (cr=425 pr=0 pw=0 time=58902 us)
 200000             HASH JOIN  (cr=425 pr=0 pw=0 time=600761 us)
   1000              TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=36 us)
 200000              INDEX FAST FULL SCAN SALES_PRODID_IDEX (cr=416 pr=0 pw=0 time=200073 us)(object id 732957)
 197303           VIEW  VW_NSO_2 (cr=425 pr=0 pw=0 time=197775 us)
 197303            HASH JOIN  (cr=425 pr=0 pw=0 time=467 us)
    990             TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=46 us)
 200000             INDEX FAST FULL SCAN SALES_PRODID_IDEX (cr=416 pr=0 pw=0 time=12 us)(object id 732957)
    800          TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=32 us)
 199907         TABLE ACCESS FULL SALES (cr=1829 pr=0 pw=0 time=42095 us)
      1          SORT AGGREGATE (cr=919 pr=0 pw=0 time=42057 us)
  42234           HASH JOIN  (cr=919 pr=0 pw=0 time=42562 us)
    300            TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=328 us)
 200000            TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=7 us)
  60000         TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=60040 us)
  30000        TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=30047 us)
 190000       VIEW  VW_NSO_1 (cr=839 pr=1540 pw=420 time=311905 us)
 190000        HASH JOIN  (cr=839 pr=1540 pw=420 time=121903 us)
  50000         TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=32 us)
 200000         INDEX FAST FULL SCAN SALES_CUSTID_IDEX (cr=432 pr=0 pw=0 time=33 us)(object id 732956)
  50000      TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=50026 us)
  70488     VIEW  VW_NSO_3 (cr=919 pr=0 pw=0 time=141556 us)
  70488      HASH JOIN  (cr=919 pr=0 pw=0 time=71067 us)
    500       TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=522 us)
 200000       TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=13 us)
```

Explanation:

	Baseline	Experiment #11	% Change	Performance Result
Count	1965	1962	-0.15%	Better than the baseline
CPU	2.08	0.93	-55%	Better than the baseline
Elapsed	5.06	1.84	-64%	Better than the baseline
Disk	12530	7332	-41%	Better than the baseline
Query	10968	5676	-48%	Better than the baseline
Current	3	8	167%	Worse than the baseline
Rows	29384	29384	0%	Same as the baseline

We found that by using this hint, the CPU time is 55% better than the baseline and the elapsed time is 64% better than the baseline. The disk and query are improved a lot as well. The query opened only 5676 blocks to get the results. However, the current has increased a lot.

Experiment #12: Using Hint 3

In this experiment, I combined the hint from experiment #11 and experiment #10 and see whether the performance can be better.

Trace Result

```
SELECT /*+ PARALLEL_INDEX INDEX(p product_pk) */ c.custid,s.custid,p.prodid,
    c.first_name,c.surname,s.sellprice,c.address,s.qty,p.price
    FROM product p, sales s, customer c
    WHERE s.prodid=p.prodid AND s.custid=c.custid
    AND c.first_name = ANY (SELECT c.first_name FROM customer c
        WHERE c.first_name like 'C%' OR c.first_name like 'G%' OR c.first_name like 'F%'
        OR c.first_name like 'E%' OR c.first_name like 'J%')
    AND c.surname = ANY (SELECT c.surname FROM customer c
        WHERE c.surname like 'H%' OR c.surname like 'D%' OR c.surname like 'B%'
        OR c.surname like 'L%' OR c.surname like 'F%')
    AND s.sellprice > (SELECT min(s.sellprice) FROM product p, sales s
        WHERE p.prodid= s.prodid
        AND (p.pname like 'P%' OR p.pname like 'M%' OR p.pname like 'K%'))
    AND c.address = ANY (SELECT c.address
        FROM customer c, sales s
        WHERE c.custid= s.custid
        AND (c.address='AUCKLAND' OR c.address='WELLINGTON' OR c.address='NAPIER'))
    AND p.prodid = ANY (SELECT p.prodid FROM product p, sales s
        WHERE p.prodid= s.prodid AND p.price>2)
    AND s.qty = ANY (SELECT s.qty FROM product p, sales s
        WHERE p.prodid= s.prodid
        AND (p.pdesc like 'D2%' OR p.pdesc like 'D3%' OR p.pdesc like 'D5%'
        OR p.pdesc like 'D8%' OR p.pdesc like 'D7%'))
    AND p.pdesc = ANY (SELECT p.pdesc FROM product p
        WHERE p.pdesc like 'D2%' OR p.pdesc like 'D3%' OR p.pdesc like
        'D5%' OR p.pdesc like 'D8%' OR p.pdesc like 'D7%' OR p.pdesc like 'D9%'
        OR p.pdesc like 'D1%')
    AND p.price > (SELECT min(p.price) FROM product p, sales s
        WHERE p.prodid= s.prodid)
    ORDER BY p.prodid,s.saleid,c.custid, c.address

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch     1960      0.98       1.90       7332       5676          8       29384
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     1962      0.99       1.90       7332       5676          8       29384

Misses in library cache during parse: 0
Optimizer mode: ALL_ROWS
Parsing user id: 10435

Rows     Row Source Operation
-------  ---------------------------------------------------
  29384  SORT ORDER BY (cr=5676 pr=7332 pw=5225 time=1886357 us)
  29384   HASH JOIN SEMI (cr=5676 pr=7048 pw=4941 time=1723383 us)
  29384    HASH JOIN SEMI (cr=4757 pr=6643 pw=4536 time=1672639 us)
  29384     HASH JOIN SEMI (cr=4350 pr=6167 pw=4060 time=1535080 us)
  29384      HASH JOIN SEMI (cr=3511 pr=2968 pw=2968 time=1187354 us)
 169362       HASH JOIN  (cr=3104 pr=1232 pw=1232 time=822548 us)
 169362        HASH JOIN  (cr=2697 pr=0 pw=0 time=376992 us)
    791         HASH JOIN SEMI (cr=868 pr=0 pw=0 time=171405 us)
    990          HASH JOIN SEMI (cr=859 pr=0 pw=0 time=157903 us)
    996           TABLE ACCESS FULL PRODUCT (cr=434 pr=0 pw=0 time=83752 us)
      1           SORT AGGREGATE (cr=425 pr=0 pw=0 time=82670 us)
 200000            HASH JOIN  (cr=425 pr=0 pw=0 time=600462 us)
   1000             TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=15 us)
 200000             INDEX FAST FULL SCAN SALES_PRODID_IDEX (cr=416 pr=0 pw=0 time=200022 us)(object id 732957)
 197303          VIEW  VW_NSO_2 (cr=425 pr=0 pw=0 time=395052 us)
 197303           HASH JOIN  (cr=425 pr=0 pw=0 time=445 us)
    990            TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=25 us)
 200000            INDEX FAST FULL SCAN SALES_PRODID_IDEX (cr=416 pr=0 pw=0 time=200018 us)(object id 732957)
    800         TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=823 us)
 199907        TABLE ACCESS FULL SALES (cr=1829 pr=0 pw=0 time=36252 us)
      1         SORT AGGREGATE (cr=919 pr=0 pw=0 time=36232 us)
  42234          HASH JOIN  (cr=919 pr=0 pw=0 time=42580 us)
    300           TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=24 us)
 200000           TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=6 us)
  60000       TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=60054 us)
  30000       TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=30048 us)
 190000      VIEW  VW_NSO_1 (cr=839 pr=1540 pw=420 time=340621 us)
 190000       HASH JOIN  (cr=839 pr=1540 pw=420 time=150618 us)
  50000        TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=32 us)
 200000        INDEX FAST FULL SCAN SALES_CUSTID_IDEX (cr=432 pr=0 pw=0 time=52 us)(object id 732956)
  50000     TABLE ACCESS FULL CUSTOMER (cr=407 pr=0 pw=0 time=33 us)
  70488    VIEW  VW_NSO_3 (cr=919 pr=0 pw=0 time=71174 us)
  70488     HASH JOIN  (cr=919 pr=0 pw=0 time=71171 us)
    500      TABLE ACCESS FULL PRODUCT (cr=9 pr=0 pw=0 time=29 us)
 200000      TABLE ACCESS FULL SALES (cr=910 pr=0 pw=0 time=16 us)
```

Explanation:

	Baseline	Experiment #12	% Change	Performance Result
Count	1965	1962	-0.15%	Better than the baseline
CPU	2.08	0.99	-52%	Better than the baseline
Elapsed	5.06	1.9	-62%	Better than the baseline
Disk	12530	7332	-41%	Better than the baseline
Query	10968	5676	-48%	Better than the baseline
Current	3	8	167%	Worse than the baseline
Rows	29384	29384	0%	Same as the baseline

	Experiment #10 using hint product_pk	Experiment #11 using hint PARALLEL_INDEX	Experiment #12 using two hints – product_pk and PARALLEL_INDEX
Count	1962	1962	1962
CPU	0.91	0.93	0.99
Elapsed	2.03	1.84	1.9
Disk	6066	7332	7332
Query	5683	5676	5676
Current	9	8	8
Rows	29384	29384	29384

We found that by using two hints – parallel_index and product_pk, the CPU time is 52% better than the baseline and the elapsed time is 62% better than the baseline. The disk and query are improved a lot as well. The query opened only 5676 blocks to get the results. Compared with only using one hint, the overall performance is about the same as the performance of experiment#10 and experiment#11.

Conclusion:

After conducting all these experiments, the best solution for this query is to first create an index on primary keys and foreign keys, and then swap table orders to make a table with smaller numbers of rows as the driving table. After that, we need to use the hint – parallel_index. With this solution, the overall elapsed time of this query can improve by 64%.

6. Reflection on Oracle's Query Optimiser

Oracle's query optimiser is software that is used to find out the best execution plan as quickly as possible. The optimiser generates all possible access paths and compares the costs of all the ways of reading the table and then chooses the plan to execute the query with the least cost and best performance. In Experiment #1, when I created indexes on all the primary keys and foreign keys for the tables, the optimiser did not use all the indexes I created and only used part of it to conduct an execution plan with better performance. It compared all the plans of reading the tables and found that the plan only using part of the indexes has the best performance. With precise statistics, the optimiser can do a good job of finding an access path with less cost. Since the query reads a high portion of the tables, the indexes might not work very well. After I conducted Experiment #2 to force the optimizer to use more hints and did not result in a better performance than only using part of the indexes. Using an unreasonable hint, the optimizer will not consider this hint. For example, in Experiment #4, we created indexes on all the columns and used the hint to force the optimiser to use all the indexes we created, the optimiser still decided to perform a full table scan on some operations because it thinks that following the hint and using all the indexes will not be an efficient way. We can also guide the optimiser to generate a better execution plan. For example, we swap the table orders and use the reasonable hint parallel_index to guide the optimiser to better performance.