Ammar Sajdi
PALCO / RealSoft

To-Index or not to INDEX

TO INDEX OR NOT TO INDEX, this is the question

Consider a table called TRANS which has four fields
ACC_NO, AMOUNT,TR_TYPE, LOCATION

LOCATION is non-uniquely indexed
ACC_NO is non-uniquely indexed

TRANS table contains 19211 records

```
SQL>  select count(*) , location from trans
      Group by location;

  COUNT (*) LOCATION
---------- ----------
     12373          0
      6453          1
       384          7
         1         99
```

12373 records match the search criteria, location=0 or 64%
6532 records match the selection criteria, LOCATION =1 or 34%
384 records match the selection criteria, LOCATION =7 or 2%
1 record match the selection criteria, LOCATION=99

```
select *
from
 trans where location=0


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      826      0.12       0.12          0       1712          0       12373
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      828      0.12       0.12          0       1712          0       12373

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 39

Rows     Row Source Operation
-------  ---------------------------------------------------
  12373  TABLE ACCESS BY INDEX ROWID TRANS
  12374   INDEX RANGE SCAN (object id 33351)

********************************************************************************
```

```
select *
from
 trans where location=1


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      432      0.02       0.02          0        908          0        6453
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      434      0.02       0.02          0        908          0        6453

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 39

Rows     Row Source Operation
-------  ---------------------------------------------------
   6453  TABLE ACCESS BY INDEX ROWID TRANS
   6454   INDEX RANGE SCAN (object id 33351)

********************************************************************************

select *
from
 trans where location=7


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       27      0.01       0.01          0         61          0         384
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       29      0.01       0.01          0         61          0         384

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 39

Rows     Row Source Operation
-------  ---------------------------------------------------
    384  TABLE ACCESS BY INDEX ROWID TRANS
    385   INDEX RANGE SCAN (object id 33351)

********************************************************************************

select *
from
 trans where location = 99


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.01       0.01          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.00          0          3          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.01       0.01          0          3          0           1

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 39

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  TABLE ACCESS BY INDEX ROWID TRANS
      2   INDEX RANGE SCAN (object id 33351)

********************************************************************************
```

```
select /*+full(trans)*/ *
from
 trans where location = 1


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      432      0.00       0.00          0        481          4        6453
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      434      0.00       0.00          0        481          4        6453

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 39

Rows     Row Source Operation
-------  ---------------------------------------------------
   6453  TABLE ACCESS FULL TRANS

********************************************************************************

select /*+full(trans)*/ *
from
 trans where location = 0


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      826      0.10       0.10          0        873          4       12373
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      828      0.10       0.10          0        873          4       12373

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 39

Rows     Row Source Operation
-------  ---------------------------------------------------
  12373  TABLE ACCESS FULL TRANS

********************************************************************************

select /*+full(trans)*/ *
from
 trans where location = 7


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       27      0.00       0.01          0         76          4         384
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       29      0.00       0.01          0         76          4         384

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 39

Rows     Row Source Operation
-------  ---------------------------------------------------
    384  TABLE ACCESS FULL TRANS
```

```
********************************************************************************

select /*+full(trans)*/ *
from
 trans where location = 99


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        2      0.02       0.02          0         51          4           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.02       0.02          0         51          4           1

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 39

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  TABLE ACCESS FULL TRANS

********************************************************************************
```

summary

Where condition	Blocks	Execution plan
Location=		
Location=		
Location=		
Location=		
Location=		
Location=		

Tuning Joins

There are several mechanism that Oracle engine can adopt to execute joins. In this section, an attempt is made to explain the most common types of joins and their effect on the overall performance on the system.

TYPES OF JOINS

1) Nested Loops
2) Hash Join
3) Sort Merge

NESTED LOOPS

In this type of joins, Oracle engine scans records from one table first one record at a time; this operation is similar to a loop iterating the

records of that particular table. This loop operation is known as the OUTER LOOP. For each record fetched using the outer loop, there is a matching record fetched from the other table making up the join. The matching record(s) is found by looping through the records of the other table.

Assume that you have two tables, Dept and Emp having the following data

DEPT

Deptno	Dname
10	Amman
20	Zarqa
30	Aqaba

EMP

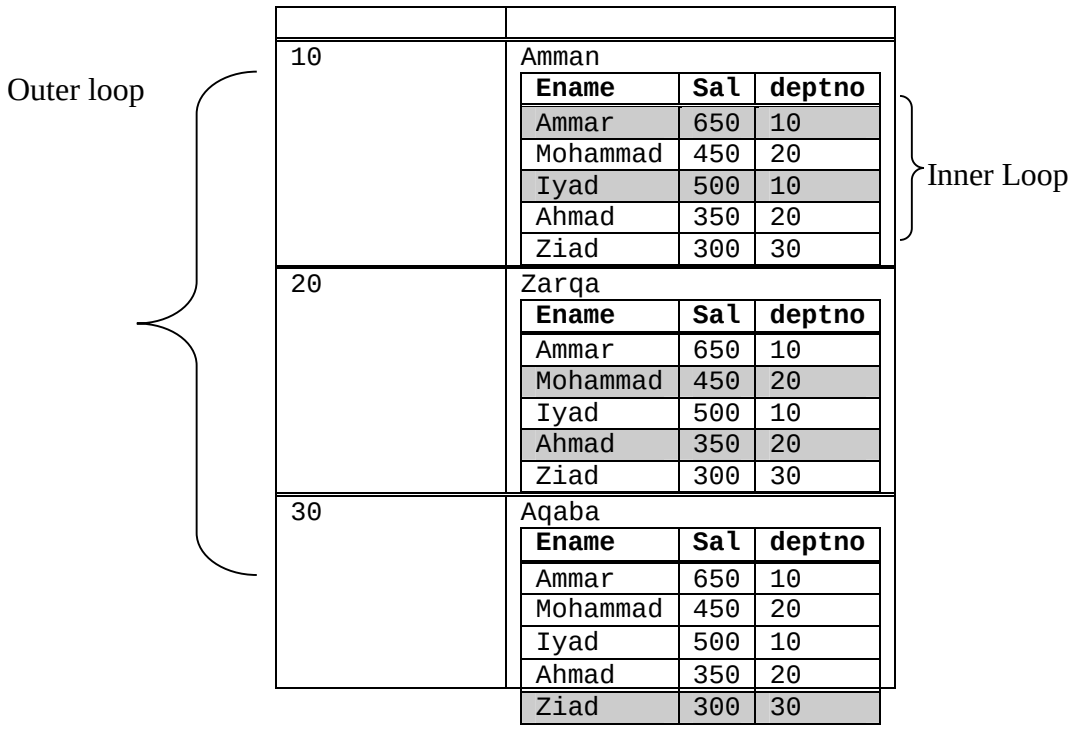
Ename	Sal	deptno
Ammar	650	10
Mohammad	450	20
Iyad	500	10
Ahmad	350	20
Ziad	300	30

The following algorithm more or less explains how the join is executed

```
LOOP   through DEPT records    <Outer>
       Fetch record from DEPT table (deptno=10)

       LOOP through EMP record  <INNER >
            Fetch record with Matching Deptno  (deptno)
       End Loop  <Inner>  -- Repeat for another matching record

End LOOP <Outer> -- repeat for the other records in the DEPT
```



	Ziad	300	30

In the above example the outer-loop is executed 3 times, once for each row. For the first row (Deptno=10), the inner loop is scans the emp table looking for a matching record (deptno=10). It fetches two records Ammar Record and Iyad record shown shaded. The second iteration of the outer loop fetch deptno=20 and the inner loop is executed several times to locate the matching records for deptno=20 and it locates two records; Mohammad and the other for Ahmad etc .. The process keeps repeating until all the the records in the outer loop are fetched.

Note: The exact number of times the inner loop will execute depends on whether the inner table (the EMP in this case) has an index on deptno or not. Clearly, if such index does not exist, the inner loop will have to execute a full scan on the emp table to find the matching records. This will be repeated for each record in the outer loop

Note: The table that is associated with the outer loop is called the DRIVING table and likewise, the table associated with the inner loop is called the DRIVEN table.

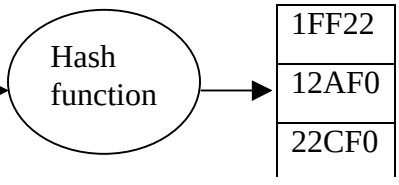
HASH JOIN

Hash joins are used for joining large data sets.
The Oracle server performs a full table scan on both tables.
One of the two-tables/data sources is used to build a hash table on the join key in memory. It then scans the larger table, looking up the hash table to find the joined rows using the hash value.
Hash Join gives superior performance if the hashed table resides in memory. If it is not possible to allocate enough memory for the hashed table, then parts of the hashed table is swapped to disk.

- Dept Table is read using full table scan.
- The records are hashed using a hash function. Each records in the Dept table translates to a hash value by applying a hash function.
- These hash values are called hash table
- Memory in allocated to hold the hash table. The memory is divided into smaller sections called partition.
- Let us assume that Oracle allocates only two partition in memory.

Dept

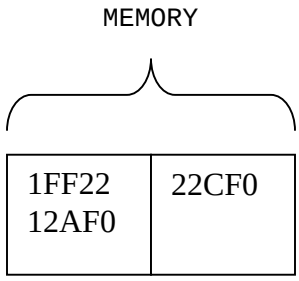
Deptno	Dname
10	Amman
20	Zarqa
30	Aqaba



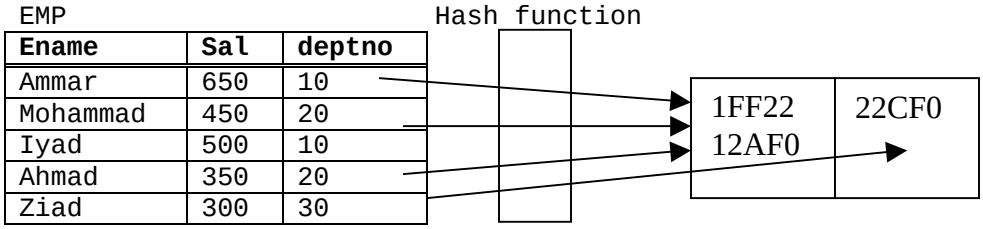
Corresponding Hash Values

1FF22
12AF0
22CF0

The following diagram shows hash values residing in the paritions



Oracle then performs a full table scan on the EMP table. For each fetched record, the hash function is applied again. A hash value results which maps to partition where the matching record resides.



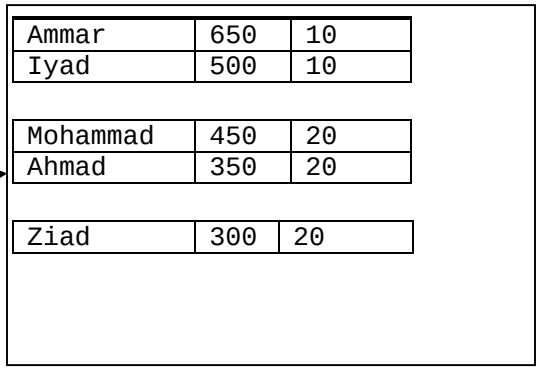
If there is not enough memory to hold the partitions of the hash tables, the partitions are swapped to disk

SORT MERGE

In this type of join, Oracle performs full table scan on each of the tables participating in a join and sort them at the same time. After the sort is complete, it becomes each to Oracle to merge matching records

EMP table

Ename	Sal	deptno
Ammar	650	10
Mohammad	450	20
Iyad	500	10
Ahmad	350	20
Ziad	300	30



As shown above, the records of the EMP are sorted into separate sort sections. If DEPT table is sorted in the same manner then the section that corresponds to Deptno=10 in the dept table is merged with the section that corresponds to deptno=10 in the EMP table

After briefly discussing the mechanism involved in executing the join operations, it is important to understand how each performs.

Performance of Join Operations

In order to understand how the above mentioned join operation perform, several cases will be tested.

The following describes the tables that will used to illustrate the our study

```
SQL>      desc sc_master
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------
---
 MTR_TRT_CODE                              NOT NULL NUMBER(38)
 MTR_NO                                    NOT NULL NUMBER(38)
 MTR_YEAR                                  NOT NULL NUMBER(38)
 MTR_CASH_CREDIT                                    VARCHAR2(20)
 MTR_CUR_CODE                              NOT NULL NUMBER(38)
 MTR_STT_CODE                              NOT NULL NUMBER(38)
 MTR_STR_NO                                NOT NULL NUMBER(38)
 MTR_DATE                                  NOT NULL DATE
 MTR_CONFIRM_FLAG                          NOT NULL VARCHAR2(1)
 MTR_CUS_SBL_NO                                     NUMBER(38)

SQL> select count(*) from sc_master;

  COUNT(*)
----------
     18015



SQL> desc sc_detail
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------
--------
 DTR_MTR_TRT_CODE                          NOT NULL NUMBER(38)
 DTR_MTR_NO                                NOT NULL NUMBER(38)
 DTR_MTR_YEAR                              NOT NULL NUMBER(38)
 DTR_SB_STR_NO                             NOT NULL NUMBER(38)
 DTR_STT_CODE                              NOT NULL NUMBER(38)
 DTR_SB_BAT_ITM_SUP_CODE                   NOT NULL VARCHAR2(4)
 DTR_SB_BAT_ITM_NO                         NOT NULL VARCHAR2(8)
 DTR_SB_BAT_NO                             NOT NULL VARCHAR2(10)
 DTR_DATE                                  NOT NULL DATE
```

```
 DTR_QTY                                   NOT NULL NUMBER(10)

SQL> select count(*) from sc_Detail;

  COUNT(*)
----------
     31279


  SQL> select  mtr_cus_sbl_no, sum(dtr_qty)
       from sc_detail d, sc_master m
       where m.mtr_Trt_Code = d.dtr_mtr_trt_code
       and   m.mtr_no       = d.dtr_mtr_no
       and   m.mtr_year     = d.dtr_mtr_year
     group by mtr_cus_sbl_no

1385 rows selected.

Elapsed: 00:00:00.81

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   SORT (GROUP BY)
   2    1     NESTED LOOPS
   3    2       TABLE ACCESS (FULL) OF 'SC_MASTER'
   4    2       TABLE ACCESS (BY INDEX ROWID) OF 'SC_DETAIL'
   5    4         INDEX (RANGE SCAN) OF 'IND_DETAIL' (NON-UNIQUE)



Statistics
----------------------------------------------------------
          8  recursive calls
         13  db block gets
      47611  consistent gets
         33  physical reads
          0  redo size
      40716  bytes sent via SQL*Net to client
      10642  bytes received via SQL*Net from client
         94  SQL*Net roundtrips to/from client
          0  sorts (memory)
          1  sorts (disk)
       1385  rows processed
```

Reverse the Order of the Tables in the FROM clause

```
SQL>    select  mtr_cus_sbl_no, sum(dtr_qty)
        from sc_master m ,sc_detail d
        where m.mtr_Trt_Code = d.dtr_mtr_trt_code
        and   m.mtr_no       = d.dtr_mtr_no
        and   m.mtr_year     = d.dtr_mtr_year
      group by mtr_cus_sbl_no
SQL> /
```

```
1385 rows selected.

Elapsed: 00:00:01.42

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   SORT (GROUP BY)
   2    1     NESTED LOOPS
   3    2       TABLE ACCESS (FULL) OF 'SC_DETAIL'
   4    2       TABLE ACCESS (BY INDEX ROWID) OF 'SC_MASTER'
   5    4         INDEX (RANGE SCAN) OF 'IND_MASTER' (NON-UNIQUE)



Statistics
----------------------------------------------------------
          8  recursive calls
         13  db block gets
      94277  consistent gets
         33  physical reads
          0  redo size
      40716  bytes sent via SQL*Net to client
      10642  bytes received via SQL*Net from client
         94  SQL*Net roundtrips to/from client
          0  sorts (memory)
          1  sorts (disk)
       1385  rows processed
```

Control the order by using ORDERED hint

```
  select /*+ USE_NL (d) ORDERED  */  mtr_cus_sbl_no, sum(dtr_qty)
      from sc_master m ,sc_detail d
      where m.mtr_Trt_Code = d.dtr_mtr_trt_code
      and   m.mtr_no       = d.dtr_mtr_no
      and   m.mtr_year     = d.dtr_mtr_year
      group by mtr_cus_sbl_no
SQL> /

1385 rows selected.

Elapsed: 00:00:00.92

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   SORT (GROUP BY) (Cost=30260 Card=426 Bytes=44304)
   2    1     NESTED LOOPS (Cost=30251 Card=426 Bytes=44304)
   3    2       TABLE ACCESS (FULL) OF 'SC_MASTER'
   4    2       TABLE ACCESS (BY INDEX ROWID) OF 'SC_DETAIL'
   5    4         INDEX (RANGE SCAN) OF 'IND_DETAIL' (NON-UNIQUE)
```

```
Statistics
----------------------------------------------------------
          8  recursive calls
         13  db block gets
      47611  consistent gets
         33  physical reads
          0  redo size
      40716  bytes sent via SQL*Net to client
      10642  bytes received via SQL*Net from client
         94  SQL*Net roundtrips to/from client
          2  sorts (memory)
          1  sorts (disk)
       1385  rows processed
```

HASH - JOIN

```
SQL> select /*+ USE_HASH (d) ordered   */  mtr_cus_sbl_no, sum(dtr_qty)
        from sc_master m ,sc_detail d
        where m.mtr_Trt_Code = d.dtr_mtr_trt_code
        and   m.mtr_no       = d.dtr_mtr_no
        and   m.mtr_year     = d.dtr_mtr_year
     group by mtr_cus_sbl_no

SQL> /

1385 rows selected.

Elapsed: 00:00:01.72

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   SORT (GROUP BY)
   2    1     HASH JOIN
   3    2       TABLE ACCESS (FULL) OF 'SC_MASTER'
   4    2       TABLE ACCESS (FULL) OF 'SC_DETAIL'



Statistics
----------------------------------------------------------
         16  recursive calls
         16  db block gets
        536  consistent gets
        282  physical reads
          0  redo size
      40716  bytes sent via SQL*Net to client
      10642  bytes received via SQL*Net from client
         94  SQL*Net roundtrips to/from client
          2  sorts (memory)
          1  sorts (disk)
       1385  rows processed
```

REVERSE The Order

```
  SQL> select /*+ USE_HASH (m) ordered*/  mtr_cus_sbl_no, sum(dtr_qty)
       from sc_detail d, sc_master m
       where m.mtr_Trt_Code = d.dtr_mtr_trt_code
       and   m.mtr_no       = d.dtr_mtr_no
       and   m.mtr_year     = d.dtr_mtr_year
      group by mtr_cus_sbl_no



1385 rows selected.

Elapsed: 00:00:01.22

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   SORT (GROUP BY)
   2    1     HASH JOIN
   3    2       TABLE ACCESS (FULL) OF 'SC_DETAIL'
   4    2       TABLE ACCESS (FULL) OF 'SC_MASTER'



Statistics
----------------------------------------------------------
         16  recursive calls
         16  db block gets
        536  consistent gets
        172  physical reads
          0  redo size
      40716  bytes sent via SQL*Net to client
      10642  bytes received via SQL*Net from client
         94  SQL*Net roundtrips to/from client
          2  sorts (memory)
          1  sorts (disk)
       1385  rows processed
```

Affecting the performance by allocating space for the HASHING function

```
SQL> alter session set HASH_AREA_SIZE=1000000;
SQL> alter session set SORT_AREA_SIZE=2000000;

Session altered.

SQL> select /*+ USE_HASH (d) ORDERED  */  mtr_cus_sbl_no, sum(dtr_qty)
  2      from sc_master m ,sc_detail d
  3      where m.mtr_Trt_Code = d.dtr_mtr_trt_code
```

```
  4      and   m.mtr_no       = d.dtr_mtr_no
  5      and   m.mtr_year     = d.dtr_mtr_year
  6     group by mtr_cus_sbl_no;

1385 rows selected.

Elapsed: 00:00:00.30


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   SORT (GROUP BY)
   2    1     HASH JOIN
   3    2       TABLE ACCESS (FULL) OF 'SC_MASTER'
   4    2       TABLE ACCESS (FULL) OF 'SC_DETAIL'



Statistics
----------------------------------------------------------
          0  recursive calls
          8  db block gets
        530  consistent gets
          0  physical reads
          0  redo size
      40716  bytes sent via SQL*Net to client
      10642  bytes received via SQL*Net from client
         94  SQL*Net roundtrips to/from client
          3  sorts (memory)
          0  sorts (disk)



The saving in the elapsed time is (1.72 minutes to  0.01)
```

SORT MERGE

```
SQL>select /*+ USE_merge (d) ORDERED  */  mtr_cus_sbl_no, sum(dtr_qty)
        from sc_master m ,sc_detail d
        where m.mtr_Trt_Code = d.dtr_mtr_trt_code
        and   m.mtr_no       = d.dtr_mtr_no
        and   m.mtr_year     = d.dtr_mtr_year
        group by mtr_cus_sbl_no

1385 rows selected.

Elapsed: 00:00:01.22

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   SORT (GROUP BY)
   2    1     MERGE JOIN
```

```
   3    2       SORT (JOIN)
   4    3         TABLE ACCESS (FULL) OF 'SC_MASTER'
   5    2       SORT (JOIN)
   6    5         TABLE ACCESS (FULL) OF 'SC_DETAIL'



Statistics
----------------------------------------------------------
         16  recursive calls
         15  db block gets
        536  consistent gets
        231  physical reads
          0  redo size
      40716  bytes sent via SQL*Net to client
      10642  bytes received via SQL*Net from client
         94  SQL*Net roundtrips to/from client
          2  sorts (memory)
          1  sorts (disk)
       1385  rows processed


SQL> alter session set sort_Area_size = 2000000;

Session altered.

SQL> get x
  1  select /*+ USE_merge (d) ORDERED  */  mtr_cus_sbl_no, sum(dtr_qty)
  2      from sc_master m ,sc_detail d
  3      where m.mtr_Trt_Code = d.dtr_mtr_trt_code
  4      and   m.mtr_no       = d.dtr_mtr_no
  5      and   m.mtr_year     = d.dtr_mtr_year
  6*    group by mtr_cus_sbl_no
SQL> /

1385 rows selected.

Elapsed: 00:00:00.41



SQL> alter session set sort_area_size=3000000;

Session altered.

SQL> get x
  1  select /*+ USE_merge (d) ORDERED  */  mtr_cus_sbl_no, sum(dtr_qty)
  2      from sc_master m ,sc_detail d
  3      where m.mtr_Trt_Code = d.dtr_mtr_trt_code
  4      and   m.mtr_no       = d.dtr_mtr_no
  5      and   m.mtr_year     = d.dtr_mtr_year
  6*    group by mtr_cus_sbl_no
SQL> /
```

```
1385 rows selected.

Elapsed: 00:00:00.11


ANOTHER APPROACH

CREATE  OR REPLACE  FUNCTION GET_FUNCTION (X1 NUMBER, X2 NUMBER, X3
NUMBER) RETURN NUMBER IS
   Y  NUMBER;
  BEGIN
    SELECT SUM(DTR_QTY) INTO Y FROM SC_DETAIL
    WHERE   DTR_MTR_YEAR =X1
    AND     DTR_MTR_TRT_CODE = X2
    AND     DTR_MTR_NO = X3;
    RETURN(Y);
 END;
SQL> /

Function created.

SQL> SELECT  mtr_cus_sbl_no  , GET_FUNCTION (MTR_NO, MTR_TRT_CODE,
MTR_YEAR) FROM SC_MASTER

18015 rows selected.

Elapsed: 00:00:03.06

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (FULL) OF 'SC_MASTER'



Statistics
----------------------------------------------------------
      18015  recursive calls
          4  db block gets
      37404  consistent gets
          0  physical reads
          0  redo size
     893749  bytes sent via SQL*Net to client
     133630  bytes received via SQL*Net from client
       1202  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
      18015  rows processed
```