M O N A S H U N I V E R S I T Y

A U S T R A L I A

School of Business Systems

PARALLEL EXECUTION IN ORACLE

- Report -

PETER XU
ID: 12288624
EMAIL: pxu1@student.monash.edu.au
PHONE: 0412310003

Semester 1, 2001

Masters by Project
(BUS4580 and BUS4590)

Supervisor: **DR. DAVID TANIAR**

COPYRIGHT NOTES

ACKNOWLEDGEMENTS

I would like to acknowledge the individual academic staffs that assisted in this project, especially Dr. David Taniar who led and supervised this project from start to finish and have assisted in the overall project. Thank you.

Peter Xu

TABLE OF CONTENTS

CHAPTER 1 - INTRODUCTION

About Parallelism

The amount of space databases take up these days can usually be enormous due to the information that is required by companies. They require this information for research and development and for the benefit of the company as a whole as they try to gain market share by knowing what their customers want and need and try to deliver their needs to the market. This is only one of the possibilities of why the databases can be large and thus easily understood why it can sometimes take a big chunk of time just to search for some small information within a database.
In this case, searching through SQL databases can be quite time consuming if we are not careful of how we approach the query.

Query optimisation can be viewed as the first step in improving the performance of queries and improve the response time in which the computer hardware computes. This optimisation, however, can be ignored when it is performed under limited number of data sets mainly due to the fact that the duration of the query will not be affected with the use of simple and direct/linear methods. This can be drastically changed for the worst if a simple query (where it is best performed via single degree) is done via a multi-degree approach.

This degree of parallelism will affect the performance of queries no matter if it is performed on small or large databases. It will affect more on larger databases rather than small because the structure of the query will be greatly affected by the approach of the degree of parallelism.

Parallel Processing

So what is parallel processing? Why is it needed? Why not just use a faster computer to speed up things? There is a simple answer to these questions and it lies largely in the laws of physics.

Parallel processing is the process of taking a large task and instead of feeding the computer this large task which may take a long time to complete, the task is divided into several smaller digestible chunks and then working on each of those smaller tasks simultaneously. This divide-and-conquer approach is to ultimately aim at completing a large task in less time than it would have if it were processed in one large task as a whole.

Since computers were invented, they were intended to solve problems fasters than a human being could. Up to now, people want computers to do more and more and to do it faster. The design of the computers have now become more complex than ever before, and with the improved circuitry design, improved instruction sets and improved algorithms to meet the demand for faster response times, it is only possible due to the advances in engineering.

Even with the advances in engineering these complex fast computers, there are speed limitations. The processing speed of processors depends on the transmission speed of information between the electronic components within the processor. Believe it or not, but the speed of these transmission is actually limited by the speed of light. Now, you may wonder how does speed of light have anything to do with computers? Well the answer is that, due to the advances in technology, the speed at which the information travels through a cable is less than the speed of light. Right now the speed of the information being transferred is reaching up to the speed of light but cannot achieve due to the laws of physics amongst the cables. With speeds of processors passed the 1GHz milestone; there is not much improvement that can be made without the use of optical communication or fibre optics. This fibre optics enables the speed of light transmission between the components of the computer. Another factor however is the density of the transistors within a processor; it can be pushed only to a certain limit. Beyond that limit, the transistors create electromagnetic interference for one another.

The limitations to the speed of processor's have resulted the hardware designers looking for another alternative to increase performance. Parallelism is the result of those efforts. Parallelism enables multiple processors to work simultaneously on several parts of a task in order to complete it faster than could be done otherwise.

So, why parallel processing and not try other methods of increasing speed? As discussed earlier, the hardware aspect of things is already to the optimised level, thus, parallel processing is the alternative to increasing speed. Parallel processing not only increases processing power, it also offers several other advantages such as Higher throughput, more fault tolerance and better price for performance, when it is implemented properly.

Now that we have addressed the need for parallel processing, here is a summary of the issues that are driving the increasing use of parallel processing in database environments:

- The need for increased speed or performance: Database sizes are increasing, queries are becoming more complex especially in data warehouse systems and the database software must somehow cope with the increasing demands that result from this complexity.
- The need for scalability: This requirement goes hand-in-hand with performance. Databases often grow rapidly, and companies need a way to easily and cost-effectively scale their systems to match that growth.
- The need for high availability: High availability refers to the need to keep a database up and running with minimal or no downtime. With the increasing use of the Internet, companies need to accommodate uses all around the clock.

How parallel Execution works

SQL statements are mostly transparent to the end users when Parallel execution is implemented. SQL statements are divided into multiple smaller units, each of which is executed by a separate process. When parallel execution is used, the user's shadow process takes on the role of the parallel coordinator. The parallel coordinator is also referred to as parallel execution coordinator or query coordinator. The parallel coordinator does the following:

1. Dynamically divides the work into smaller units that can be parallelised.
2. Acquires a sufficient number of parallel processes to execute the individual smaller units. These parallel processes are called parallel execution server processes and also parallel slave processes and slave processes.
3. Assigns each unit of work to a slave process.
4. Collects and combines the results from the slave processes and return those results to the user process.
5. Release the slave processes after the work is done.

In Oracle, there are many operations that can be parallelised, below is a list of the operations:

List of Oracle Paralellable operations.

The Oracle server can use parallel execution for any of these operations:

- Table scan
- Nested loop join
- Sort merge join
- Hash join
- "Not in"
- Group by
- Select distinct
- Union and union all
- Aggregation
- PL/SQL functions called from SQL
- Order by
- Create table as select
- Create index
- Rebuild index
- Rebuild index partition
- Move partition
- Split partition
- Update
- Delete
- Insert ... select
- Enable constraint (the table scan is parallelised)
- Star transformation

In this research, a few of the mentioned operations will be examined in detail and the affect of the degree of parallelism will be shown to demonstrate the parallelism and the ways that parallelism can be achieved. The investigation is focused on the *degree of parallelism* of the commands or queries and different methods of approach will affect the speed and efficiency of the final result. Through this investigation, the improvement or advantages of introducing parallelism to selected commands or queries form the focus of this research assignment.

Operations Covered in this report.

Chapter 2: CREATE table / Index statements
- Showing Parallel DDL

Chapter 3: SELECT statements
- Showing the degree of parallelism

Chapter 4: SELECT DISCTINCT and/or ORDER BY
1. Showing sorting and in which level it performs the operation.
- The degree of parallelism affected in search.

Chapter 5: JOIN statements
- Showing the join statements
    - Simple join statement:
      SELECT
      FROM…,…,…
      WHERE…;
    - Nested Joins:
      SELECT
      FROM…
      WHERE…(SELECT …);

Chapter 6: GROUP BY statements
- SELECT …
  FROM…
  WHERE…
  GROUP BY …
- Demonstrated through single table operations, and
- Through multiple table operations.

Chapter 7: DML
- UPDATE,
- INSERT,
- DELETE.

Chapter 8: Extended Utilities
- PARALLEL loading
- Parallel recovering / duplication.

Parallel Execution of these queries makes use of the divide-and-conquer method. It divides a task among multiple processes in order to complete the task faster. This allows Oracle to get that advantage of multiple CPUs on a machine. The parallel processes acting on behalf of a single task are called parallel slave processes. In this research we will be looking mainly on the ways that these slave processes can be used to increase the performance of queries.

CHAPTER 2 - CREATION OF TABLES.

With creation of tables, the normal clauses are quite well known. They are done without any extra clauses and thus uses a standardised default format.

For example, a basic method of creating a table without any means of speeding up the process for later query execution, this is how it would be done.

```
CREATE TABLE customer(
custid       NUMBER(5)           CONSTRAINT customer_custid_pk PRIMARY KEY,
last         VARCHAR2(20)        NOT NULL,
first        VARCHAR2(20)        NOT NULL,
MI           VARCHAR2(1),
cadd         VARCHAR2(30)        NOT NULL,
city         VARCHAR2(30)        NOT NULL,
state        VARCHAR2(2)         NOT NULL,
zip          VARCHAR2(10)        NOT NULL,
dphone       VARCHAR2(10),
ephone       VARCHAR2(10))
```

To check for the level of parallelism of this table, a SQL statement can be used. If we follow the above table creation with another statement to check the degree of parallelism, we can do the following clause. This clause can be used anywhere to check for a table’s use of parallelism.

```
SELECT * FROM user_tables
WHERE table_name = ‘<table name>’;
```

Thus, for our example this is the finding that resulted:

```
SQL> SELECT DEGREE FROM user_tables
  2  WHERE table_name = 'CUSTOMER';

DEGREE
----------
         1
```

We find that the default or nominal degree of parallelism is set to one (1). This one process thus runs in serial rather than parallel. If the query was executed under a very heavily loaded database with over 100meg worth of data, there will be dramatic level of delay noticeable to the human eye.

On the other hand, this creation of tables/indexes can make use of the parallelism option in Oracle, reducing the amount of turn around time for the query. This is done via the Parallel clause of the create table and create index statements.

The different ways of creating a table will be investigated; the following is a list of the attempted trials:

- Table creation in parallel.
- Index Creation in Parallel.
- Serially Created Index.
- Parallel Created Index.

2.1 Table Creation

Table creation in parallel.

For example, the creation of *customer* table above would be the same, except that an extra clause is placed after the last statement. This clause overrides the default settings of parallelism for this table alone. An example of this is of below:

```
CREATE TABLE customer(
custid       NUMBER(5)          CONSTRAINT customer_custid_pk PRIMARY KEY,
last         VARCHAR2(20)       NOT NULL,
first        VARCHAR2(20)       NOT NULL,
MI           VARCHAR2(1),
cadd         VARCHAR2(30)       NOT NULL,
city         VARCHAR2(30)       NOT NULL,
state        VARCHAR2(2)        NOT NULL,
zip          VARCHAR2(10)       NOT NULL,
dphone       VARCHAR2(10),
ephone       VARCHAR2(10))
PARALLEL (DEGREE 4);
```

Index Creation in Parallel.

Multiple processes can work together simultaneously to create an index. By dividing the work necessary to create an index among multiple server processes, the Oracle Server can create the index more quickly than if a single server process creates the index sequentially.

Parallel index creation works in much the same way as a table scan with an ORDER BY clause. The table is randomly sampled and a set of index keys is found that equally divides the index into the same number of pieces as the degree of parallelism.

Parallel local index creation uses a single server set. Each server process in the set is assigned a table partition to scan, and to build an index partition for. Because half as many server processes are used for a given degree of parallelism, parallel local index creation can be run with a higher degree of parallelism.

The PARALLEL clause in the CREATE INDEX command is the only way in which you can specify the degree of parallelism for creating the index. If the degree of parallelism is not specified along with the parallel clause of CREATE INDEX, then the number of CPUs is used as the degree of parallelism. If there is no parallel clause, index creation will be done serially.

Serially Created Index.

```
SQL> CREATE INDEX cust_idx1 ON customer(first, custid);

Index created.
```

Parallel Created Index.

```
SQL> CREATE INDEX cust_idx1 ON customer(first, custid)
  2  PARALLEL (degree 5);

Index created.
```

2.2 Table Alterations

Altering tables

Having put the parallel clause in the creation table line, an alternative is to modify the table structure so that it can be processed in parallel. To do this, we use the ALTER command.

For example, if the table is either in serial or there is a need to change the degree of parallelism, then the following can be used to make this change.

```
SQL> ALTER TABLE customer PARALLEL (degree 8);

Table altered.
```

To change the parallelism back to serial processing, the following operation is performed.

```
SQL> ALTER TABLE customer PARALLEL (degree 1);

Table altered.
```

Degree one (1) means that it will use one processor for this query and thus performed serially.

Altering tables will be the next experimentation, and this will be the list of the tried attempts:

- Index parallelism

Index parallelism:

```
SQL> ALTER INDEX cust_idx1 PARALLEL (degree 2);

Table altered.
```

The same thing with Index, the parallelism can be reverted back to serial processing by doing the following:

```
SQL> ALTER INDEX cust_idx1 PARALLEL (degree 1);

Table altered.
```

CHAPTER 3 - SELECT STATEMENTS

Select statements are the first and most basic level of specifying the degree of parallelism by using hints or by using a PARALLEL clause. Parallel and parallel_index hints are used to specify the degree of parallelism used for queries. From using a normal SELECT statement without any clauses, the default serial processing will be engaged, where as if a parallel clause is present, a query may be sped up depending on the size of the database being searched.

There are many ways in which parallel execution can be applied. The following demonstrates some of the ways possible for such execution.

In normal situations without any extra clauses, the following is the method used. This is assuming that no parallel clauses were used during the process of creating the table.

Below is a list of the attempted queries which demonstrates this:

- Normal Select
- Parallel Select
- Aggregate SELECT statements
- Parallel Hint on Primary Key
- LIKE clause
- Parallel Like
- LESS THAN clause
- Parallel LESS THAN clause
- LESS THAN and OR clause
- Parallel LESS THAN and OR clause
- LESS THAN clause using only the primary key.

3.1 SELECT

Normal Select

```
SQL> SELECT DEGREE FROM user_tables
  2  WHERE table_name = 'ITEM';

DEGREE
----------
         1
```

Thus a normal select statement would be,

```
SQL> SELECT * FROM item;

   ITEMID ITEMDESC                                                CATEGORY
--------- -------------------------------------------------- ---------------
----------
      894 Women's Hiking Shorts                Women's Clothing
      897 Women's Fleece Pullover              Women's Clothing
      995 Children's Beachcomber Sandals              Children's Clothing
      559 Men's Expedition Parka               Men's Clothing
      786 3-Season Tent                        Outdoor Gear
```

The same results can be obtained in parallel by using the following clause. The PARALLEL clause in the statement overrides any default or assigned value for the table. Thus, shown below, shows a select statement with PARALLEL (item, 4). This literally means that the table is to be searched in parallel of the order of 4^{th} degree.

Parallel Select

```
SQL> SELECT /*+ PARALLEL(item, 4) */ * FROM item;

   ITEMID ITEMDESC                                                 CATEGORY
--------- -------------------------------------------------- ---------------
----------
      894 Women's Hiking Shorts                                          Women's
Clothing
      897 Women's Fleece Pullover                                        Women's
Clothing
      995 Children's Beachcomber Sandals                                 Children's
Clothing
      559 Men's Expedition Parka                                         Men's
Clothing
      786 3-Season Tent                                                  Outdoor
Gear
```

If we use the V$PQ_TQSTAT view, which is available within the ORACLE view table pool, the level of parallelism can be shown and proven that the query is actually doing some work.

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER     TQ_ID SERVER_TYP PROCESS     NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   QC                30      1113
         1         0 Producer   P003               0        24
         1         0 Producer   P002               0        24
         1         0 Producer   P001               0        24
         1         0 Producer   P000              30      1041
```

Above illustrates the parallel slave processes act as producers, and the query coordinator acts as the consumer. It shows four (4) producers (4^{th} degree) working on the query where in fact, since it is such as small table, the producer P000 does all the processing.

Other possible examples of the statement consist of the *aggregate* functions, where multiple clauses hone down to a smaller, more precise portions of the query.

3.2 AGGREGATE

Aggregate SELECT statements

```
SQL> SELECT count(*) AS "Student Count",
  2         avg(age) AS "Average Age"
  3  FROM student;

Student Count Average Age
------------- -----------
           20        36.7
```

The above query gets a total COUNT of the record number of students in the table and the does an average calculation of the AGE.
The clauses used here are COUNT(*), which counts the number of records (rows) and AVG(age) calculates the average age of the overall number of students.

Then, showing the parallelism used in the above query,

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER     TQ_ID SERVER_TYP PROCESS     NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   QC                30      1113
         1         0 Producer   P003               0        24
         1         0 Producer   P002               0        24
         1         0 Producer   P001               0        24
         1         0 Producer   P000              30      1041
```

Showing that the query is using only one process or one level of parallelism.

Thus for example, the following is an illustration to show the HINT clause to force the user of parallelism and the proof of the use.

```
SQL> SELECT /*+ PARALLEL (student, 4) */
  2   count(*) AS "Student Count",
  3   avg(age) AS "Average Age"
  4  FROM student;

Student Count Average Age
------------- -----------
           20        36.7
```

And the proof:

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER     TQ_ID SERVER_TYP PROCESS     NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   QC                 5       306
         1         0 Producer   P003               1        66
         1         0 Producer   P002               1        66
         1         0 Producer   P001               1        66
         1         0 Producer   P000               2       108
```

The above indicates the use of each of the four (4) producers P000 – P003, each of them does some kind of work and thus make use of the 4th degree of parallelism.

```
SQL> SELECT /*+ PARALLEL (item, 4) */ * FROM item
  2  WHERE category = 'Outdoor Gear';

   ITEMID ITEMDESC                                          CATEGORY
--------- -------------------------------------------------- ---------------
----------
      786 3-Season Tent                                     Outdoor Gear
```

The above shows the forced PARALLEL clause for a query that is a non-primary key.

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER     TQ_ID SERVER_TYP PROCESS     NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   QC                 1       130
         1         0 Producer   P003               0        24
         1         0 Producer   P002               0        24
         1         0 Producer   P001               0        24
         1         0 Producer   P000               1        58
```

As shown above four (4) processes are used to produce the results. It is queried on a non-primary key, which results in using the parallelism specified.

3.3 PARALLEL HINTS

Parallel Hint on Primary Key

If the same example was used on the actual *primary key*, Oracle takes this as a different situation and a different result is produced.

```
SQL> SELECT /*+ PARALLEL (item, 4) */ * FROM item
  2  where itemid = 907;

   ITEMID ITEMDESC                                          CATEGORY
--------- -------------------------------------------------- ---------------
----------
      907 Hard Hat                                          Men's Clothing
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER     TQ_ID SERVER_TYP PROCESS     NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   QC                 1       130
         1         0 Producer   P003               0        24
         1         0 Producer   P002               0        24
         1         0 Producer   P001               0        24
         1         0 Producer   P000               1        58
```

As shown above, the results are of the previous query (the one that was a non-primary key). Because the results were of the previous query, the actual query with Parallel Hint does not really produce any significant on the level of parallelism, thus no record of parallelism usage.

3.4 LIKE

LIKE clause:

The LIKE clause makes use of the "%" signs to indicate everything else. Specifying some character between these "%" allows the ORACLE to search for anything with what was specified. This is very useful when searching for a part of a word within other words.

```
SQL> select * from student
  2  where fname like '%e%';

      SID FNAME                LNAME                      AGE HCOLOR
HEIGHT
--------- -------------------- -------------------- --------- ---------- ---
------
       10 mel                  hack                        12 green
190
       11 jones                Ng                          23 black
180
       13 pete                 flang                       24 black
188
       16 james                long                        15 brown
155
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

Since there is no parallelism specified in the SELECT command, there is no record of parallel usage.

Parallel Like

```
SQL> select /*+ PARALLEL(student, 3) */ * FROM student
  2  WHERE fname LIKE '%e%';

      SID FNAME                   LNAME                          AGE HCOLOR
HEIGHT
--------- -------------------- -------------------- --------- ---------- ---
------
       10 mel                   hack                            12 green
190
       11 jones                 Ng                              23 black
180
       13 pete                  flang                           24 black
188
       16 james                 long                            15 brown
155
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER     TQ_ID SERVER_TYP PROCESS      NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   QC                 4       200
         1         0 Producer   P002               0        24
         1         0 Producer   P001               0        24
         1         0 Producer   P000               4       152
```

Four (4) rows returned by the query using a PARALLEL hint. The recorded illustrates parallelism usage of 4 processes and the four rows actually is produced by the first process and thus does not make use of the other two due to the small size of the table.

3.5 LESS THAN

LESS THAN clause

The LESS THAN clause uses the “<” as a symbol as a LESS THAN clause. It compares the value on the right with the one on the left and returns true if it satisfies the condition.

```
SQL> SELECT * FROM student
  2  WHERE age < 20;

      SID FNAME                   LNAME                          AGE HCOLOR
HEIGHT
--------- -------------------- -------------------- --------- ---------- ---
------
        1 bob                   smart                           15 black
170
       10 mel                   hack                            12 green
190
       16 james                 long                            15 brown
155
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

The above illustrates a LESS THAN clause in the query. It does not make use of any parallelism thus ORACLE does not record any parallel usage.

Parallel LESS THAN clause

```
SQL> SELECT /*+ PARALLEL(student, 3) */ * from student
  2  where age < 20;

      SID FNAME                LNAME                      AGE HCOLOR
HEIGHT
--------- -------------------- -------------------- --------- ---------- ---
------
        1 bob                  smart                       15 black
170
       10 mel                  hack                        12 green
190
       16 james                long                        15 brown
155
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   QC                  3        168
         1          0 Producer   P002                0         24
         1          0 Producer   P001                0         24
         1          0 Producer   P000                3        120
```

The above illustrates a LESS THAN clause with a Parallel Hint. It, again, uses only the one process and returns the result to the consumer process.

3.6 OR

LESS THAN and OR clause

```
SQL> SELECT * from student
  2  where age < 20 or fname like '%e%';

      SID FNAME                LNAME                      AGE HCOLOR
HEIGHT
--------- -------------------- -------------------- --------- ---------- ---
------
        1 bob                  smart                       15 black
170
       10 mel                  hack                        12 green
190
       11 jones                Ng                          23 black
180
       13 pete                 flang                       24 black
188
       16 james                long                        15 brown
155
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

LESS THAN and OR clause with Parallelism

```
SQL> SELECT /*+ PARALLEL(student, 3) */ * from student
  2  where age < 20 or fname like '%e%';

      SID FNAME                    LNAME                          AGE HCOLOR
HEIGHT
--------- ------------------- ------------------- --------- ---------- ---
------
        1 bob                       smart                          15  black
170
       10 mel                       hack                           12  green
190
       11 jones                    Ng                              23  black
180
       13 pete                      flang                          24  black
188
       16 james                    long                            15  brown
155
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER     TQ_ID SERVER_TYP PROCESS     NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   QC                 5       232
         1         0 Producer   P002               0        24
         1         0 Producer   P001               0        24
         1         0 Producer   P000               5       184
```

The above queries illustrate the parallelism being specified, but they only use the one process thus proves to be no use to the overall purpose.

The only query that proved to be making use of the parallelism is the one below where two joint clauses AVG() was used. The reason mainly because to calculate the average, Oracle has to add up all the columns to get the total, then count the number of rows it used to add up the total, and then do another calculation to generate the average (total/number of rows).

Thus producing the following:

```
DFO_NUMBER     TQ_ID SERVER_TYP PROCESS     NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   QC                 5       306
         1         0 Producer   P003               1        66
         1         0 Producer   P002               1        66
         1         0 Producer   P001               1        66
         1         0 Producer   P000               2       108
```

LESS THAN and OR clause

```
SQL> SELECT /*+ PARALLEL(student, 3) */ * from student
  2  WHERE sid < 10 OR sid = 19;

      SID FNAME                    LNAME                          AGE HCOLOR
HEIGHT
--------- -------------------- -------------------- --------- ---------- ---
------
        1 bob                      smart                           15 black
170
        2 sam                      taps                            21 blond
160
        3 craig                    stone                           34 brown
150
        4 tom                       spat                           66 red
166
        5 pat                      stone                           45 brown
179
        6 tim                      cray                            55 black
166
        7 paul                     craz                            77 grey
187
        8 dawn                     mal                             66 brown
130
        9 jack                     jones                           55 blond
180
       19 wang                      chris                          21 grey
155

10 rows selected.
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER     TQ_ID SERVER_TYP PROCESS      NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   QC                10       388
         1         0 Producer   P002               0        24
         1         0 Producer   P001               0        24
         1         0 Producer   P000              10       340
```

As shown above, by using the OR clause, the parallel hint is actually used where as if the query was based upon just the primary key, there is not parallelism used as this column is already indexed by default.

LESS THAN clause using only the primary key.

```
SQL> SELECT /*+ PARALLEL(student, 3) */ * from student
  2  WHERE sid < 10;

      SID FNAME                    LNAME                          AGE  HCOLOR
HEIGHT
--------- -------------------- -------------------- --------- ---------- ---
------
        1 bob                  smart                           15  black
170
        2 sam                  taps                            21  blond
160
        3 craig                stone                           34  brown
150
        4 tom                   spat                           66  red
166
        5 pat                  stone                           45  brown
179
        6 tim                  cray                            55  black
166
        7 paul                 craz                            77  grey
187
        8 dawn                 mal                             66  brown
130
        9 jack                 jones                           55  blond
180

9 rows selected.
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

As illustrated above, there is no parallelism used even if the parallel hint was used. Thus the parallel hint in this situation is void due to the fact that the primary key is already indexed.

3.7 PLAN TABLE

Plan table

The EXPLAIN PLAN command displays the execution plan chosen by the Oracle optimiser for SELECT, UPDATE, INSERT, and DELETE statements. A statement's execution plan is the sequence of operations that Oracle performs to execute the statement. By examining the execution plan, you can see exactly how Oracle executes the SQL statement.

Before you can issue an EXPLAIN PLAN statement, you must create a table to hold its output.

```
CREATE TABLE plan_table
   (statement_id          VARCHAR2(30),
    timestamp             DATE,
    remarks               VARCHAR2(80),
    operation             VARCHAR2(30),
    options               VARCHAR2(30),
    object_node           VARCHAR2(128),
    object_owner          VARCHAR2(30),
    object_name           VARCHAR2(30),
    object_instance       NUMERIC,
    object_type           VARCHAR2(30),
    optimizer             VARCHAR2(255),
    search_columns   NUMERIC,
    id                    NUMERIC,
    parent_id             NUMERIC,
    position              NUMERIC,
    cost                  NUMERIC,
    cardinality           NUMERIC,
    bytes                 NUMERIC,
    other_tag             VARCHAR2(255),
    other                 LONG);
```

Now that the plan table has been established, the prior queries will experimented using this extra tool, and together with the V$PQ_TQSTAT statistic table provided by table it is then possible to show how ORACLE processes each of the queries.

3.8 SELECT

SELECT STATEMENTS

Select statements selects the data in rows and columns from one or more tables.

Below is a list of the experimented queries, which demonstrate these select statements.

- Normal Select
- Parallel Select
- Aggregate SELECT statements
- Parallel Aggregate SELECT statements
- Parallel Clause on NON-primary key
- Parallel Hint on Primary Key
- Like Clause
- Parallel LIKE
- LESS THAN clause
- Parallel LESS THAN clause
- LESS THAN and OR clause – No parallelism

```
SQL> select degree from user_tables
  2  where table_name = 'ITEM';

DEGREE
----------
         1
```

```
SQL> SELECT * FROM item;

   ITEMID ITEMDESC                                          CATEGORY
--------- -------------------------------------------------- ---------------
----------
      894 Women's Hiking Shorts                    Women's Clothing
      897 Women's Fleece Pullover                  Women's Clothing
      995 Children's Beachcomber Sandals                  Children's Clothing
      559 Men's Expedition Parka                   Men's Clothing
      786 3-Season Tent                            Outdoor Gear
```

```
SQL> explain plan for
  2  SELECT * FROM item;

Explained.
```

```
SQL> select operation
  2  from plan_table;

OPERATION
------------------------------
SELECT STATEMENT
TABLE ACCESS
```

As shown above, the simple select statement simply has one access to the table to get the query result.

Parallel Select

```
SQL> SELECT /*+ PARALLEL(item, 4) */ * FROM item;

   ITEMID ITEMDESC                                                   CATEGORY
--------- -------------------------------------------------- ---------------
----------
      894 Women's Hiking Shorts                                           Women's
Clothing
      897 Women's Fleece Pullover                                         Women's
Clothing
      995 Children's Beachcomber Sandals                                  Children's
Clothing
      559 Men's Expedition Parka                                          Men's
Clothing
      786 3-Season Tent                                                   Outdoor
Gear
```

```
SQL> explain plan for
  2  SELECT /*+ PARALLEL(item, 4) */ * FROM item;

Explained.
```

```
SQL> select operation
  2  from plan_table;

OPERATION
------------------------------
SELECT STATEMENT
TABLE ACCESS
```

As shown above, it has the same table access' comparing the Normal SELECT and the Parallel SELECT.

If we use the V$PQ_TQSTAT view, which is available within the ORACLE view table pool, the level of parallelism can be shown and proven that the query is actually doing some work.

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   QC                30      1113
         1         0 Producer   P003               0        24
         1         0 Producer   P002               0        24
         1         0 Producer   P001               0        24
         1         0 Producer   P000              30      1041
```

The above illustrates the parallel slave processes act as producers, and the query coordinator acts as the consumer. It shows four (4) producers (4th degree) working on the query where in fact, since it is such as small table, the producer P000 does all the processing. This includes both the TABLE ACCESS and the SELECT statement.

3.9 AGGREGATE

Aggregate SELECT statements

```
SQL> SELECT count(*) AS "Student Count",
  2         avg(age) AS "Average Age"
  3  FROM student;

Student Count Average Age
------------- -----------
           20        36.7
```

The above query gets a total COUNT of the record number of students in the table and the does an average calculation of the AGE.
The clauses used here are COUNT(*), which counts the number of records (rows) and AVG(age) calculates the average age of the overall number of students.

```
SQL> explain plan for
  2  SELECT count(*) AS "Student Count",
  3         avg(age) AS "Average Age"
  4  FROM student;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 SORT
        2 TABLE ACCESS

3 rows selected.
```

Three operations are being made here, as the query goes for a sort before the select statement.

Then, showing the parallelism used in the above query,

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER     TQ_ID SERVER_TYP PROCESS     NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   QC                30      1113
         1         0 Producer   P003               0        24
         1         0 Producer   P002               0        24
         1         0 Producer   P001               0        24
         1         0 Producer   P000              30      1041
```

Showing that the query is using only one process or one level of parallelism. The first process does all three of the query, including TABLE ACCESS, SORT and the final SELECT STATEMENT.

Parallel Aggregate SELECT statements

```
SQL> SELECT /*+ PARALLEL (student, 4) */
  2   count(*) AS "Student Count",
  3   avg(age) AS "Average Age"
  4  FROM student;

Student Count Average Age
------------- -----------
           20        36.7
```

```
SQL> explain plan for
  2  SELECT /*+ PARALLEL (student, 4) */
  3   count(*) AS "Student Count",
  4   avg(age) AS "Average Age" FROM student;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 SORT
        2 SORT
        3 TABLE ACCESS
```

As illustrated, the operation actually takes two sorts for the query to operate in parallel.

Using a parallel hint, the following can be observed as more processes are used for the same outcome.

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   QC                  5        306
         1          0 Producer   P003                1         66
         1          0 Producer   P002                1         66
         1          0 Producer   P001                1         66
         1          0 Producer   P000                2        108
```

The above indicates the use of each of the four (4) producers P000 – P003, each of them does some kind of work and thus make use of the 4th degree of parallelism.

Parallel Clause on NON-primary key

```
SQL> explain plan for
  2  SELECT /*+ PARALLEL (item, 4) */ * FROM item
  3  WHERE category = 'Outdoor Gear';

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 TABLE ACCESS
```

The above uses merely two accesses to display the query. The use of this actually uses the basic straight off table access followed by the select command.

Parallel Hint on Primary Key

```
SQL> explain plan for
  2  SELECT /*+ PARALLEL (item, 4) */ * FROM item
  3  where itemid = 907;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 TABLE ACCESS
        2 INDEX
```

As it was expected, this query uses an extra access to the index due to the fact that the primary key is already on default indexed.

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER     TQ_ID SERVER_TYP PROCESS      NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   QC                 1       130
         1         0 Producer   P003               0        24
         1         0 Producer   P002               0        24
         1         0 Producer   P001               0        24
         1         0 Producer   P000               1        58
```

As shown above, the results are of the previous query (the one that was a non-primary key). Because the results were of the previous query, the actual query with Parallel Hint does not really produce any significant on the level of parallelism, thus no record of parallelism usage

3.10 LIKE

Like Clause

```
SQL> select * from student
  2  where fname like '%e%';

      SID FNAME                LNAME                      AGE HCOLOR
HEIGHT
--------- -------------------- -------------------- --------- ---------- ---
------
       10 mel                  hack                        12 green
190
       11 jones                Ng                          23 black
180
       13 pete                 flang                       24 black
188
       16 james                long                        15 brown
155
```

```
SQL> explain plan for
  2  select * from student
  3  where fname like '%e%';

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 TABLE ACCESS
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

Since there is no parallelism specified in the SELECT command, there is no record of parallel usage.

Parallel LIKE

```
SQL> select /*+ PARALLEL(student, 3) */ * FROM student
  2  WHERE fname LIKE '%e%';

      SID FNAME                LNAME                      AGE HCOLOR
HEIGHT
--------- -------------------- -------------------- --------- ---------- ---
------
       10 mel                  hack                        12 green
190
       11 jones                Ng                          23 black
180
       13 pete                 flang                       24 black
188
       16 james                long                        15 brown
155
```

```
SQL> explain plan for
  2  select /*+ PARALLEL(student, 3) */ * FROM student
  3  WHERE fname LIKE '%e%';

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 TABLE ACCESS
```

The above illustrates that there were no difference between the two LIKE clauses when used in parallel or normal.

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER     TQ_ID SERVER_TYP PROCESS     NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   QC                 4       200
         1         0 Producer   P002               0        24
         1         0 Producer   P001               0        24
         1         0 Producer   P000               4       152
```

Four (4) rows returned by the query using a PARALLEL hint. The recorded illustrates parallelism usage of 4 processes and the four rows actually is produced by the first process and thus does not make use of the other two due to the small size of the table. Although there were two operations made to perform this query, there was only one process used.

3.11 LESS THAN

LESS THAN clause

```
SQL> SELECT * FROM student
  2  WHERE age < 20;

      SID FNAME                LNAME                      AGE  HCOLOR
HEIGHT
--------- -------------------- -------------------- --------- ---------- ---
------
        1 bob                   smart                      15  black
170
       10 mel                   hack                       12  green
190
       16 james                long                       15  brown
155
```

```
SQL> explain plan for
  2  SELECT * FROM student
  3  WHERE age < 20;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 TABLE ACCESS
```

The above uses the basic direct access to get the above query result.

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

The above illustrates a LESS THAN clause in the query. It does not make use of any parallelism thus ORACLE does not record any parallel usage.

Parallel LESS THAN clause

```
SQL> SELECT /*+ PARALLEL(student, 3) */ * from student
  2  where age < 20;

      SID FNAME                LNAME                      AGE  HCOLOR
HEIGHT
--------- -------------------- -------------------- --------- ---------- ---
------
        1 bob                   smart                      15  black
170
       10 mel                   hack                       12  green
190
       16 james                long                       15  brown
155
```

```
SQL> explain plan for
  2  SELECT /*+ PARALLEL(student, 3) */ * from student
  3  where age < 20;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 TABLE ACCESS
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER     TQ_ID SERVER_TYP PROCESS     NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   QC                 3       168
         1         0 Producer   P002               0        24
         1         0 Producer   P001               0        24
         1         0 Producer   P000               3       120
```

Illustrated above, is the parallelism that is used to process the above query. Although the query uses two operations, there is only one process used to produce the output. Once again, this is due to the size of the table thus only one process is used.

LESS THAN and OR clause – No parallelism

```
SQL> SELECT * from student
  2  where age < 20 or fname like '%e%';

      SID FNAME                LNAME                      AGE  HCOLOR
HEIGHT
--------- -------------------- -------------------- --------- ---------- ---
------
        1 bob                   smart                        15  black
170
       10 mel                   hack                         12  green
190
       11 jones                Ng                            23  black
180
       13 pete                  flang                        24  black
188
       16 james                long                          15  brown
155
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 TABLE ACCESS
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

The same operations as the parallel version are being performed here but yielding the same results.

CHAPTER 4 - SELECT DISTINCT

Select distinct shows the list of requested data one once only. Thus other repeated or same records with the matching characters will not be displayed. This is useful in cases where you want to know how many different types to categorise the data.

Below is a list of the experimented queries, which demonstrate this.

- WITHOUT DISTINCT on non-PK, ORDER BY
- WITHOUT DISTINCT on non-PK with Parallelism, ORDER BY
- With DISTINCT, without parallelism, ORDER BY
- Without DISTINCT on Primary Key, without parallelism
- With DISTINCT on PK, with Parallelism.
- With DISTINCT, non-PK
- With DISTINCT and parallelism
-

4.1 DISTINCT

WITHOUT DISTINCT on non-PK, ORDER BY

```
SQL> select fname from student
  2  order by fname;

FNAME
--------------------
bob
craig
dawn
jack
jack
james
john
jones
mel
pat
paul
paul
pete
rick
sam
stan
stan
tim
tom
wang

20 rows selected.
```

Here, we see duplicates in the FNAME field. Thus, if we want to see how many different names we have for this STUDENT table, we can make use of the DISTINCT clause.

```
SQL> explain plan for
  2  select fname from student
  3  order by fname;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 SELECT STATEMENT
         1 SORT
         2 TABLE ACCESS
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

As it can be seen, from the result of the plan table, and the results from the V$PQ_TQSTAT table, it is clearly shown that the query was performed serially and that this query requires three (3) operations to complete.

WITHOUT DISTINCT on non-PK with Parallelism, ORDER BY

```
SQL> select /*+ parallel (student, 4) */ fname
  2  from student
  3  order by fname;

FNAME
--------------------
bob
craig
dawn
jack
jack
james
john
jones
mel
pat
paul

FNAME
--------------------
paul
pete
rick
sam
stan
stan
tim
tom
wang

20 rows selected.
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P001                8         72
         1          0 Consumer   P000               12         99
         1          0 Producer   P004               20        141
         1          0 Producer   P002                0         24
         1          0 Ranger     QC                 20        336
         1          1 Consumer   QC                 20        165
         1          1 Producer   P001                8         69
         1          1 Producer   P000               12         96

8 rows selected.
```

```
SQL> explain plan for
  2  select /*+ parallel (student, 4) */ fname
  3  from student
  4  order by fname;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 SELECT STATEMENT
         1 SORT
         2 TABLE ACCESS
```

It can be shown that, four processes are used to produce this query result. It first takes the table and accesses the records, sorting the records according to ORDER BY clause and finally selects the required fields from the table as the output.

4.2 ORDER BY

With DISTINCT, without parallelism, ORDER BY

```
SQL> select distinct fname
  2  from student
  3  order by fname;

FNAME
--------------------
bob
craig
dawn
jack
james
john
jones
mel
pat
paul
pete

FNAME
--------------------
rick
sam
stan
tim
tom
wang

17 rows selected.
```

```
SQL> explain plan for
  2  select distinct fname
  3  from student
  4  order by fname;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 SELECT STATEMENT
         1 SORT
         2 TABLE ACCESS
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

Using a DISTINCT clause makes little difference as it can be seen. The query still requires three (3) operations to yield the result.

4.3 PRIMARY KEY

Without DISTINCT on Primary Key, without parallelism

```
SQL> SELECT DISTINCT sid FROM student;

      SID
---------
        1
        2
        3
        4
        5
        6
        7
        8
        9
       10
       11
       12
       13
       14
       15
       16
       17
       18
       19
       20

20 rows selected.
```

```
SQL> explain plan for
  2  SELECT DISTINCT sid FROM student;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 SELECT STATEMENT
         1 SORT
         2 TABLE ACCESS
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

As it can be seen, the query illustrates the same results as if it was
Since this query was made on the indexed primary key. Lets see if a parallel hint will make any difference to the result if a parallel hint was made on the primary key.

With DISTINCT on PK, with Parallelism.

```
SQL> SELECT /*+ PARALLEL(student,3) */ DISTINCT sid
  2  FROM student;

      SID
---------
        1
        3
        4
        7
        9
       10
       12
       14
       15
       16
       17
       19
        2
        5
        6
        8
       11
       13
       18
       20

20 rows selected.
```

```
SQL> explain plan for
  2  SELECT /*+ PARALLEL(student,3) */ DISTINCT sid
  3  from student;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 SELECT STATEMENT
         1 SORT
         2 TABLE ACCESS
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P001               12         96
         1          0 Consumer   P000                8         80
         1          0 Producer   P004                0         48
         1          0 Producer   P003               20        128
         1          1 Consumer   QC                 20        128
         1          1 Producer   P001               12         72
         1          1 Producer   P000                8         56
```

As shown above, the parallel hint is actually used, unlike in previous experiments, this example uses the parallelism due to the fact that the processing of the whole table was required before a choosing the individual records to be displayed, ensuring that the record was displayed only once.

4.4 NON-PRIMARY KEY

With DISTINCT, non-PK

```
SQL> SELECT DISTINCT FNAME FROM STUDENT;

FNAME
--------------------
bob
craig
dawn
jack
james
john
jones
mel
pat
paul
pete

FNAME
--------------------
rick
sam
stan
tim
tom
wang

17 rows selected.
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

```
SQL> explain plan for
  2  select distinct fname
  3  from student;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 SELECT STATEMENT
         1 TABLE ACCESS

8 rows selected.
```

The above illustrates the use of distinct without parallelism on a non-primary key. And the process that it uses in order to produce this result.

With DISTINCT and parallelism

```
SQL> SELECT /*+ PARALLEL (student, 3) */ DISTINCT fname
  2  from student;

FNAME
--------------------
bob
craig
jack
john
paul
rick
sam
tom
wang
dawn
james

FNAME
--------------------
jones
mel
pat
pete
stan
tim

17 rows selected.
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   P001              11        112
         1         0 Consumer   P000               9        101
         1         0 Producer   P004               0         48
         1         0 Producer   P003              20        165
         1         1 Consumer   QC                17        147
         1         1 Producer   P001               9         76
         1         1 Producer   P000               8         71
```

```
SQL> explain plan for
  2  SELECT /*+ PARALLEL (student, 3) */ DISTINCT fname
  3  from student;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 SELECT STATEMENT
         1 SORT
         2 TABLE ACCESS

11 rows selected.
```

CHAPTER 5 – JOIN

Join statements allow linkage of two or more tables for a query statement. To illustrate this and the different ways of performing parallelism, the following operations are performed:

- A normal Join statement to join two or more tables,
- A parallel hinted Join statement joining two or more tables
- A normal Nested Query
- A parallel hinted Nested Query
- A normal Nested Join,
- A parallel hinted nested join,
- A normal Join statement with SORTing,
- A parallel hinted statement with SORTing,
- A normal join statement with SORTing and DISTINCT clause,
- A parallel hinted statement with SORTing and DISTINCT clause.
- A nested join statement with SORTing and GREATER THAN clause.
- A parallel hinted nested join statement with SORTing and GREATER THAN clause.
- A nested join statement with SORTing and LIKE clause.
- A parallel hinted, nested join statement with SORTing and LIKE clause.
- Nested Query with IN operator
- Nested Query with IN operator with Parallelism
- Nested Query with NOT IN operator
- Nested Query with NOT IN operator with Parallelism
- Nested Query with ANY operator
- Nested Query with ANY operator and parallelism
- Nested Query with EXIST operator
- Nested Query with EXIST operator, with parallelism

5.1 JOIN

Normal Join

```
SQL> select c.first, c.last, co.orderdate, ol.order_price, ol.quantity
  2  from customer c, cust_order co, orderline ol
  3  where c.custid = co.custid AND
  4        co.orderid = ol.orderid AND
  5        c.last = 'Harris';

FIRST                LAST                 ORDERDATE ORDER_PRICE  QUANTITY
-------------------- -------------------- --------- ----------- ---------
Paula                Harris               29-MAY-01      259.99         1
```

The above links the three tables customer, cust_order and orderline choosing data from each table and displaying only the matching query according to the WHERE condition.

Analysing the processing of this query, we perform the EXPLAIN PLAN and view the v$pq_tqstat table for the necessary information.
Checking the default degree for each of the tables so that we know what degree was used for each table;
For the CUSTOMER table:

```
SQL> select degree from user_tables
  2  where table_name = 'CUSTOMER';

DEGREE
----------
         8
```

```
SQL> select degree from user_tables
  2  where table_name = 'CUST_ORDER';

DEGREE
----------
         1
```

```
SQL> select degree from user_tables
  2  where table_name = 'ORDERLINE';

DEGREE
----------
         1
```

Using degree of eight (8) for only the customer table and the rest using the default degree of one (1), the following result was recorded:

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER     TQ_ID SERVER_TYP PROCESS     NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   QC                 1       164
         1         0 Producer   P004               0        24
         1         0 Producer   P003               0        24
         1         0 Producer   P002               0        24
         1         0 Producer   P001               0        24
         1         0 Producer   P000               1        68
```

```
SQL> explain plan for
  2  select c.first, c.last, co.orderdate, ol.order_price, ol.quantity
  3  from customer c, cust_order co, orderline ol
  4  where c.custid = co.custid AND
  5        co.orderid = ol.orderid AND
  6        c.last = 'Harris';

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 NESTED LOOPS
        2 NESTED LOOPS
        3 TABLE ACCESS
        4 TABLE ACCESS
        5 TABLE ACCESS

6 rows selected.
```

By observing the results above, the operations needed to complete this query has increased. Instead of accessing only one table like previously, there are now three tables being accessed. Nested Loops are now used to find the link between the tables and then the final SELECT STATEMENT. Observing the v$pq_tqstat results, it shows the parallelism usage. It shows that ORACLE is only using the single process to complete each task breakdown. By having the CUSTOMER table as degree of eight (8), it was hoping that the v$pq_tqstat would show something different but due to the size of the tables, ORACLE only uses one process for efficiency.

Parallel Hinted Join

Because we are using a parallel hint, the default values of degree will be overwritten, thus the original values of the degree of each table can be ignored.
Here, we are using the degree of two (2) for each of the tables for this query.

```
SQL> select /*+ parallel (customer, 2, orderline, 2, cust_order, 2) */
  2  c.first, c.last, co.orderdate, ol.order_price, ol.quantity
  3  from customer c, cust_order co, orderline ol
  4  where c.custid = co.custid AND
  5  co.orderid = ol.orderid AND
  6  c.last = 'Harris';

FIRST                LAST                 ORDERDATE ORDER_PRICE  QUANTITY
-------------------- -------------------- --------- ----------- ---------
Paula                Harris               29-MAY-01      259.99
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER     TQ_ID SERVER_TYP PROCESS     NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   QC                 1       164
         1         0 Producer   P004               0        24
         1         0 Producer   P003               0        24
         1         0 Producer   P002               0        24
         1         0 Producer   P001               0        24
         1         0 Producer   P000               1        68
```

```
SQL> explain plan for
  2  select /*+ parallel (customer, 2, orderline, 2, cust_order, 2) */
  3  c.first, c.last, co.orderdate, ol.order_price, ol.quantity
  4  from customer c, cust_order co, orderline ol
  5  where c.custid = co.custid AND
  6  co.orderid = ol.orderid AND
  7  c.last = 'Harris';

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 NESTED LOOPS
        2 NESTED LOOPS
        3 TABLE ACCESS
        4 TABLE ACCESS
        5 TABLE ACCESS

6 rows selected.
```

From the above result, it differs very little from the normal join statement and the parallel hinted join statement. This is because of the fact that because they are using different tables, ORACLE automatically uses a different process for each table access.

5.2 NESTED

A normal Nested,

Here, a normal nested query is performed. It makes use of two select statements, one as an outer final select where the inner select finds the maximum height from the table and uses this figure for the outer SELECT to perform the final query.

```
SQL> select fname, height
  2  from student
  3  where height = (select max(height) from student);

FNAME                    HEIGHT
-------------------- ---------
mel                         190
```

```
SQL> explain plan for
  2  select fname, height
  3  from student
  4  where height = (select max(height) from student);

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 FILTER
        2 TABLE ACCESS
        3 SORT
        4 TABLE ACCESS
```

As it can be seen, since we are using two level of select statements, there are two table access', but they are performed one statement from a SORT operation. The first table access is the inner SELECT where it finds the maximum height from the student table. The sort operation is performed to find the maximum height for the inner select statement. The second table access is to access the student table where a FILTER is used to filter out the height of students that do not match the maximum height, and there the final select statement is made to complete the query.

A parallel hinted Nested Query

```
SQL> select /*+ parallel (student, 2) */
  2  fname, height
  3  from student
  4  where height = (select max(height) from student);

FNAME                    HEIGHT
-------------------- ---------
mel                         190
```

```
SQL> explain plan for
  2  select /*+ parallel (student, 2) */
  3  fname, height
  4  from student
  5  where height = (select max(height) from student);

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 TABLE ACCESS
        2 SORT
        3 TABLE ACCESS
```

```
DFO_NUMBER     TQ_ID SERVER_TYP PROCESS      NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   QC                 1        58
         1         0 Producer   P001               0        24
         1         0 Producer   P000               1        34
```

It can be seen here that a parallel hinted nested operation uses one less operation, the filter. This proves to show that using parallelism in the right time can actually save processing time and thus increase efficiency of the query.

A normal Nested Join

```
SQL> select c.first, co.methpmt, o.quantity
  2  from customer c, cust_order co, orderline o
  3  where c.custid = co.custid AND
  4        co.orderid = o.orderid AND
  5        o.quantity = (select max(quantity) from orderline);

FIRST                METHPMT    QUANTITY
-------------------- --------- ---------
Alissa               CC                3
```

Because of the fact that all default degree for each table is used (degree of 1) there is no record of any parallelism usage.

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

```
SQL> explain plan for
  2  select c.first, co.methpmt, o.quantity
  3  from customer c, cust_order co, orderline o
  4  where c.custid = co.custid AND
  5  co.orderid = o.orderid AND
  6  o.quantity = (select max(quantity) from orderline);

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 NESTED LOOPS
        2 NESTED LOOPS
        3 TABLE ACCESS
        4 SORT
        5 TABLE ACCESS
        6 TABLE ACCESS
        7 INDEX
        8 TABLE ACCESS
        9 INDEX

10 rows selected.
```

As it can be seen here, that the nested join requires many operations to satisfy the query. It involves indexing, nested loops and sorting to get this query, and thus proving to be a very expensive query.

A parallel hinted nested join

Three tables used in this query was modified to have a degree of 2 for the purpose of experimenting with parallel hinted nested join queries.

```
SQL> ALTER TABLE customer parallel (degree 2);

Table altered.

SQL> ALTER TABLE cust_order parallel (degree 2);

Table altered.

SQL> ALTER TABLE orderline parallel (degree 2);

Table altered.
```

```
SQL> select c.first, co.methpmt, o.quantity
  2  from customer c, cust_order co, orderline o
  3  where c.custid = co.custid AND
  4  co.orderid = o.orderid AND
  5  o.quantity = (select max(quantity) from orderline);

FIRST                METHPMT    QUANTITY
-------------------- --------- ---------
Alissa               CC                3
```

```
SQL> explain plan for
  2  select c.first, co.methpmt, o.quantity
  3  from customer c, cust_order co, orderline o
  4  where c.custid = co.custid AND
  5  co.orderid = o.orderid AND
  6  o.quantity = (select max(quantity) from orderline);

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 NESTED LOOPS
        2 NESTED LOOPS
        3 TABLE ACCESS
        4 SORT
        5 SORT
        6 TABLE ACCESS
        7 TABLE ACCESS
        8 INDEX
        9 TABLE ACCESS
       10 INDEX

11 rows selected.
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER     TQ_ID SERVER_TYP PROCESS     NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Producer   P000               1        84
         1         0 Producer   P001               0        24
```

Since the parallel hint have been default changed all the tables to 2 degree of parallelism, it can be seen from above that there were two processes used, and in this case, more operations were used to produce this query.

5.3 SORTING

A normal Join statement with SORTing,

All tables were changed back to default serial processing for this normal join statement experimentation.

```
SQL> ALTER TABLE customer parallel (degree 1);

Table altered.

SQL> ALTER TABLE cust_order parallel (degree 1);

Table altered.

SQL> ALTER TABLE orderline parallel (degree 1);

Table altered.
```

Since using sub queries, one result usually are produced. Thus for this example an extra clause of ORDER BY is used to see what the implications it has on the query.

```
SQL> select c.first, co.methpmt, o.quantity
  2  from customer c, cust_order co, orderline o
  3  where c.custid = co.custid AND
  4  co.orderid = o.orderid AND
  5  o.quantity = (select max(quantity) from orderline)
  6  order by c.first;

FIRST                METHPMT    QUANTITY
-------------------- --------- ---------
Alissa               CC                3
```

Since the query is performed serially, there is no parallelism used.

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

```
SQL> explain plan for
  2  select c.first, co.methpmt, o.quantity
  3  from customer c, cust_order co, orderline o
  4  where c.custid = co.custid AND
  5  co.orderid = o.orderid AND
  6  o.quantity = (select max(quantity) from orderline)
  7  order by c.first;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 SORT
        2 NESTED LOOPS
        3 NESTED LOOPS
        4 TABLE ACCESS
        5 SORT
        6 TABLE ACCESS
        7 TABLE ACCESS
        8 INDEX
        9 TABLE ACCESS
       10 INDEX

11 rows selected.
```

The above illustrates the wasted ORDER BY clause where it doesn't do anything for this query as only one result is only produced. The operations total however is the same as the previous parallel hinted nested join of 11 operations.
The only difference is that ID 1 has a SORT here where as the previous query on the parallel hinted nested join has an ID 1 of NESTED LOOPS.

A parallel hinted statement with SORTing,

The tables are updated with parallelism of 2 for each of the three tables.

```
SQL> ALTER TABLE customer parallel (degree 2);

Table altered.

SQL>  ALTER TABLE cust_order parallel (degree 2);

Table altered.

SQL> ALTER TABLE orderline parallel (degree 2);

Table altered.
```

```
SQL> select c.first, co.methpmt, o.quantity
  2  from customer c, cust_order co, orderline o
  3  where c.custid = co.custid AND
  4  co.orderid = o.orderid AND
  5  o.quantity = (select max(quantity) from orderline)
  6  order by c.first;

FIRST                METHPMT    QUANTITY
-------------------- --------- ---------
Alissa               CC                3
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER     TQ_ID SERVER_TYP PROCESS     NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   P001               0        27
         1         0 Consumer   P000               1        43
         1         0 Producer   P003               0        24
         1         0 Producer   P002               1        40
         1         0 Ranger     QC                 1       139
         1         1 Consumer   QC                 1        64
         1         1 Producer   P001               0        24
         1         1 Producer   P000               1        40
```

```
SQL> explain plan for
  2  select c.first, co.methpmt, o.quantity
  3  from customer c, cust_order co, orderline o
  4  where c.custid = co.custid AND
  5  co.orderid = o.orderid AND
  6  o.quantity = (select max(quantity) from orderline)
  7  order by c.first;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 SORT
        2 NESTED LOOPS
        3 NESTED LOOPS
        4 TABLE ACCESS
        5 SORT
        6 SORT
        7 TABLE ACCESS
        8 TABLE ACCESS
        9 INDEX
       10 TABLE ACCESS
       11 INDEX

12 rows selected.
```

The above shows that using parallelism in this case uses an extra operation where as before uses one less.

5.4 DISTINCT

A normal join statement with SORTing and DISTINCT clause,

The tables were changed back to their original form of being serial processing.

```
SQL> alter table customer parallel (degree 1);

Table altered.

SQL> alter table cust_order parallel (degree 1);

Table altered.

SQL> alter table orderline parallel (degree 1);

Table altered.
```

```
SQL> select distinct c.first, co.methpmt, o.quantity
  2  from customer c, cust_order co, orderline o
  3  where c.custid = co.custid AND
  4  co.orderid = o.orderid AND
  5  o.quantity = (select min(quantity) from orderline)
  6  order by c.first;

FIRST                METHPMT    QUANTITY
-------------------- --------- ---------
Alissa               CC                1
Paula                CC                1
```

```
SQL> explain plan for
  2  select distinct c.first, co.methpmt, o.quantity
  3  from customer c, cust_order co, orderline o
  4  where c.custid = co.custid AND
  5  co.orderid = o.orderid AND
  6  o.quantity = (select min(quantity) from orderline)
  7  order by c.first;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 SORT
        2 NESTED LOOPS
        3 NESTED LOOPS
        4 TABLE ACCESS
        5 SORT
        6 TABLE ACCESS
        7 TABLE ACCESS
        8 INDEX
        9 TABLE ACCESS
       10 INDEX

11 rows selected.
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

Since the query is performed under serial, there is no rows selected when viewing the v$pq_tqstat table.
Form the results above, it illustrates that many NESTED LOOPS and tables accesses are required to perform this query.

A parallel hinted, nested join statement with SORTing and DISTINCT clause.

```
SQL> select distinct c.first, co.methpmt, o.quantity
  2  from customer c, cust_order co, orderline o
  3  where c.custid = co.custid AND
  4  co.orderid = o.orderid AND
  5  o.quantity = (select min(quantity) from orderline)
  6  order by c.first;

FIRST                METHPMT    QUANTITY
-------------------- --------- ---------
Alissa               CC                1
Paula                CC                1
```

```
SQL> explain plan for
  2  select distinct c.first, co.methpmt, o.quantity
  3  from customer c, cust_order co, orderline o
  4  where c.custid = co.custid AND
  5  co.orderid = o.orderid AND
  6  o.quantity = (select min(quantity) from orderline)
  7  order by c.first;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

       ID OPERATION
--------- ------------------------------
        0 SELECT STATEMENT
        1 SORT
        2 NESTED LOOPS
        3 NESTED LOOPS
        4 TABLE ACCESS
        5 SORT
        6 SORT
        7 TABLE ACCESS
        8 TABLE ACCESS
        9 INDEX
       10 TABLE ACCESS
       11 INDEX

12 rows selected.
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER     TQ_ID SERVER_TYP PROCESS     NUM_ROWS     BYTES
---------- --------- ---------- ---------- --------- ---------
         1         0 Consumer   P001               1        42
         1         0 Consumer   P000               2        59
         1         0 Producer   P003               0        24
         1         0 Producer   P002               3        71
         1         0 Ranger     QC                 3       196
         1         1 Consumer   QC                 2        79
         1         1 Producer   P001               1        39
         1         1 Producer   P000               1        40
```

```
8 rows selected.
```

As it can be observed from the result above, the operation for this query requires one less amount of operations, however, for this query, ORACLE makes use of the parallelism of degree 2.

5.5 GREATER THAN

A nested join statement with SORTing and GREATER THAN clause.

The tables were given the original degree of 1, thus running in series.

```
SQL> alter table customer parallel (degree 1);

Table altered.

SQL> alter table cust_order parallel (degree 1);

Table altered.

SQL> alter table orderline parallel (degree 1);

Table altered.
```

The nested join statement with SORTing and GREATER THAN clause.

```
SQL> select distinct c.first, co.methpmt, o.quantity
  2  from customer c, cust_order co, orderline o
  3  where c.custid = co.custid AND
  4  co.orderid = o.orderid AND
  5  o.quantity > (select avg(quantity) from orderline)
  6  order by c.first;

FIRST                METHPMT     QUANTITY
-------------------- --------- ----------
Alissa               CC                 3
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

```
SQL> explain plan for
  2  select distinct c.first, co.methpmt, o.quantity
  3  from customer c, cust_order co, orderline o
  4  where c.custid = co.custid AND
  5  co.orderid = o.orderid AND
  6  o.quantity > (select avg(quantity) from orderline)
  7  order by c.first;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 SELECT STATEMENT
         1 SORT
         2 FILTER
         3 HASH JOIN
         4 HASH JOIN
         5 TABLE ACCESS
```

```
         6 TABLE ACCESS
         7 TABLE ACCESS
         8 SORT
         9 TABLE ACCESS

10 rows selected.
```

As it can be seen, the GREATER THAN clause created the use of HASH JOINs for this query. It has accessed the tables in four occasions with other operations such as FILTER and SORT. Let see how this changes with parallelism execution.

A parallel hinted nested join statement with SORTing and GREATER THAN clause.

The tables were given the degree of two (2) giving all three tables parallelism of two processes.

```
SQL> ALTER TABLE customer parallel (degree 2);

Table altered.

SQL> ALTER TABLE cust_order parallel (degree 2);

Table altered.

SQL> TABLE orderline parallel (degree 2);

Table altered.
```

```
SQL> select distinct c.first, co.methpmt, o.quantity
  2  from customer c, cust_order co, orderline o
  3  where c.custid = co.custid AND
  4  co.orderid = o.orderid AND
  5  o.quantity > (select avg(quantity) from orderline)
  6  order by c.first;

FIRST                METHPMT     QUANTITY
-------------------- --------- ----------
Alissa               CC                 3
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P003                1         60
         1          0 Consumer   P002                5        102
         1          0 Producer   P001                0         48
         1          0 Producer   P000                6        114
         1          1 Consumer   P003                1         62
         1          1 Consumer   P002                5        121
         1          1 Producer   P001                0         48
         1          1 Producer   P000                6        135
         1          2 Consumer   P001                2         82
         1          2 Consumer   P000                4        113
         1          2 Producer   P003                1         64
         1          2 Producer   P002                5        131
         1          3 Consumer   P001                3         75
         1          3 Consumer   P000                1         57
         1          3 Producer   P003                0         48
         1          3 Producer   P002                4         84
         1          4 Producer   P001                3         72
         1          4 Producer   P000                1         39
```

```
SQL> explain plan for
  2  select distinct c.first, co.methpmt, o.quantity
  3  from customer c, cust_order co, orderline o
  4  where c.custid = co.custid AND
  5  co.orderid = o.orderid AND
  6  o.quantity > (select avg(quantity) from orderline)
  7  order by c.first;

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 SELECT STATEMENT
         1 SORT
         2 FILTER
         3 HASH JOIN
         4 HASH JOIN
         5 TABLE ACCESS
         6 TABLE ACCESS
         7 TABLE ACCESS
         8 SORT
         9 SORT
        10 TABLE ACCESS

11 rows selected.
```

Comparing the normal serial access and the parallel access, it can be seen that for parallel execution of this query, an extra SORT is required to complete the operation.

A nested join statement with SORTing and LIKE clause.

The tables are returned back to the serial execution method.

```
SQL> ALTER TABLE customer parallel (degree 1);

Table altered.

SQL> alter table cust_order parallel (degree 1);

Table altered.

SQL> alter table orderline parallel (degree 1);

Table altered.
```

```
SQL> select distinct c.first, co.methpmt, o.quantity
  2  from customer c, cust_order co, orderline o
  3  where c.custid = co.custid AND
  4  co.orderid = o.orderid AND
  5  co.methpmt LIKE (select distinct methpmt from cust_order
  6  where methpmt = 'CC');

FIRST                METHPMT     QUANTITY
-------------------- --------- ----------
Alissa               CC                 1
Alissa               CC                 3
Paula                CC                 1
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

```
SQL> explain plan for
  2  select distinct c.first, co.methpmt, o.quantity
  3  from customer c, cust_order co, orderline o
  4  where c.custid = co.custid AND
  5  co.orderid = o.orderid AND
  6  co.methpmt LIKE (select distinct methpmt from cust_order
  7  where methpmt = 'CC');

Explained.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 SELECT STATEMENT
         1 SORT
         2 FILTER
         3 HASH JOIN
         4 HASH JOIN
         5 TABLE ACCESS
         6 TABLE ACCESS
         7 TABLE ACCESS
         8 SORT
         9 TABLE ACCESS

10 rows selected.
```

Using the LIKE clause in this JOIN statement creates a similar execution style to the previous nested join statement with SORTing and GREATER THAN clause. They both have the same operations and both were running in serial. Lets see if this is the case with the parallel execution method.

5.6 LIKE

A parallel hinted, nested join statement with SORTing and LIKE clause.

The tables where given a parallelism of two (2).

```
SQL> ALTER TABLE customer parallel (degree 2);

Table altered.

SQL> ALTER TABLE cust_order parallel (degree 2);

Table altered.

SQL> ALTER TABLE orderline parallel (degree 2);

Table altered.
```

```
SQL> select distinct c.first, co.methpmt, o.quantity
  2  from customer c, cust_order co, orderline o
  3  where c.custid = co.custid AND
  4  co.orderid = o.orderid AND
  5  co.methpmt LIKE (select distinct methpmt from cust_order
  6  where methpmt = 'CC');

FIRST                METHPMT     QUANTITY
-------------------- --------- ----------
Alissa               CC                 1
Alissa               CC                 3
Paula                CC                 1
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P003                1         62
         1          0 Consumer   P002                5        121
         1          0 Producer   P001                0         48
         1          0 Producer   P000                6        135
         1          1 Consumer   P003                1         60
         1          1 Consumer   P002                5        102
         1          1 Producer   P001                0         48
         1          1 Producer   P000                6        114
         1          2 Consumer   P001                2         82
         1          2 Consumer   P000                4        113
         1          2 Producer   P003                1         64
         1          2 Producer   P002                5        131
         1          3 Consumer   P001                3         75
         1          3 Consumer   P000                1         57
         1          3 Producer   P003                0         48
         1          3 Producer   P002                4         84
         1          4 Consumer   QC                  4        111
         1          4 Producer   P001                3         72
         1          4 Producer   P000                1         39

19 rows selected.
```

By observing the results of the number of processes that were used, it shows that by joining three tables together and if there was a parallel hints, each table is given a process each in which to operate on. Hence the above, P000 to P003 as there were four tables that were access during this query; three on the normal select and one on the nested select.

5.7 IN

Nested Query with IN operator

The In Operator in this example lists the first name, last name and the item name of customers whose first name is also in the student table.

The tables are given parallel degree of one (1), which is running in serial.

```
SQL> ALTER TABLE customer parallel (degree 1);

Table altered.

SQL> alter table cust_order parallel (degree 1);

Table altered.

SQL> alter table orderline parallel (degree 1);

Table altered.
```

```
SQL> select c.first, c.last, i.itemdesc
  2  from customer c, item i, inventory inv, cust_order co, orderline ol
  3  where c.custid = co.custid AND
  4  co.orderid = ol.orderid AND
  5  ol.invid = inv.invid AND
  6  inv.itemid = i.itemid AND
  7  c.first IN (select fname from student);

FIRST                LAST               ITEMDESC
-------------------- -------------------- -----------------------------------
----------
Paula                Harris                   3-Season Tent
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

Since there is no parallelism used, the above doesn't yield any results.

```
SQL> explain plan for
  2  select c.first, c.last, i.itemdesc
  3  from customer c, item i, inventory inv, cust_order co, orderline ol
  4  where c.custid = co.custid AND
  5  co.orderid = ol.orderid AND
  6  ol.invid = inv.invid AND
  7  inv.itemid = i.itemid AND
  8  c.first IN (select fname from student);

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                              COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                         13
         1 HASH JOIN                                13
         2 HASH JOIN                                11
         3 HASH JOIN                                 9
         4 HASH JOIN                                 7
         5 HASH JOIN                                 5
         6 TABLE ACCESS                              1
         7 VIEW                                      3
         8 SORT                                      3
         9 TABLE ACCESS                              1
        10 TABLE ACCESS                              1

        ID OPERATION                              COST
---------- ------------------------------ ----------
        11 TABLE ACCESS                              1
        12 TABLE ACCESS                              1
        13 TABLE ACCESS                              1

14 rows selected.
```

The above illustrates the cost of the IN operation. It makes use of many HASH JOINs where the most costly operations were performed.

Nested Query with IN operator with Parallelism

The tables where given a parallelism of two (2).

```
SQL> ALTER TABLE customer parallel (degree 2);

Table altered.

SQL> ALTER TABLE cust_order parallel (degree 2);

Table altered.

SQL> ALTER TABLE orderline parallel (degree 2);

Table altered.
```

```
SQL> select c.first, c.last, i.itemdesc
  2  from customer c, item i, inventory inv, cust_order co, orderline ol
  3  where c.custid = co.custid AND
  4  co.orderid = ol.orderid AND
  5  ol.invid = inv.invid AND
  6  inv.itemid = i.itemid AND
  7  c.first IN (select fname from student);

FIRST                LAST              ITEMDESC
-------------------- -------------------- ----------------------------------
----------
Paula                Harris                  3-Season Tent
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P001                9         76
         1          0 Consumer   P000                9         78
         1          0 Producer   QC                 18        154
         1          1 Consumer   P003                6         90
         1          1 Consumer   P002                2         46
         1          1 Producer   QC                  8        136
         1          2 Consumer   P001               20        359
         1          2 Consumer   P000               10        200
         1          2 Producer   QC                 30        559
         1          3 Consumer   P001                0         48
         1          3 Consumer   P000                6        165
         1          3 Producer   P003                0         48
         1          3 Producer   P002                6        165
         1          4 Consumer   P003                1         72
         1          4 Consumer   P002                1         75
         1          4 Producer   P001                0         48
         1          4 Producer   P000                2         99
         1          5 Consumer   P003                1         58
         1          5 Consumer   P002                5         98
         1          5 Producer   P001                0         48
         1          5 Producer   P000                6        108
         1          6 Consumer   P001                0         48
         1          6 Consumer   P000                1         75
         1          6 Producer   P003                0         48
         1          6 Producer   P002                1         75
         1          7 Consumer   P001                3         81
         1          7 Consumer   P000                1         59
         1          7 Producer   P003                0         48
         1          7 Producer   P002                4         92
         1          8 Consumer   P003                1         76
         1          8 Consumer   P002                0         48
         1          8 Producer   P001                0         48
         1          8 Producer   P000                1         76
         1          9 Consumer   P001                0         48
         1          9 Consumer   P000                1         75
         1          9 Producer   P003                1         75
         1          9 Producer   P002                0         48
         1         10 Consumer   QC                  1         85
         1         10 Producer   P001                0         24
         1         10 Producer   P000                1         61

40 rows selected.
```

```
SQL> EXPLAIN PLAN FOR
  2  select c.first, c.last, i.itemdesc
  3  from customer c, item i, inventory inv, cust_order co, orderline ol
  4  where c.custid = co.custid AND
  5  co.orderid = ol.orderid AND
  6  ol.invid = inv.invid AND
  7  inv.itemid = i.itemid AND
  8  c.first IN (select fname from student);

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ---------------------------- ----------
         0 SELECT STATEMENT                       13
         1 HASH JOIN                              13
         2 HASH JOIN                              11
         3 HASH JOIN                               9
         4 HASH JOIN                               7
         5 HASH JOIN                               5
         6 TABLE ACCESS                            1
         7 VIEW                                    3
         8 SORT                                    3
         9 TABLE ACCESS                            1
        10 TABLE ACCESS                            1

        ID OPERATION                            COST
---------- ---------------------------- ----------
        11 TABLE ACCESS                            1
        12 TABLE ACCESS                            1
        13 TABLE ACCESS                            1

14 rows selected.
```

From the above, it can be seen that there is the same costs between the serial and the parallel implementation methods, however, there the more processes used, and there may be a speed improvement, but because there is an increase in machine usage, that is the only down side. But usually if the usage is the same, and there is an improvement in speed, then that is the optimum goal.

5.8 NOT IN

Nested Query with NOT IN operator

The tables are given parallel degree of one (1), which is running in serial.

```
SQL> ALTER TABLE customer parallel (degree 1);

Table altered.

SQL> alter table cust_order parallel (degree 1);

Table altered.

SQL> alter table orderline parallel (degree 1);

Table altered.
```

```
SQL> select c.first, c.last, i.itemdesc
  2  from customer c, item i, inventory inv, cust_order co, orderline ol
  3  where c.custid = co.custid AND
  4  co.orderid = ol.orderid AND
  5  ol.invid = inv.invid AND
  6  inv.itemid = i.itemid AND
  7  c.first NOT IN (select fname from student);

FIRST                LAST                 ITEMDESC
-------------------- -------------------- -----------------------------------
Alissa               Chang                3-Season Tent
Alissa               Chang                Women's Hiking Shorts
Alissa               Chang                Women's Hiking Shorts
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

Since there is no parallelism used, the above doesn't yield any results.

```
SQL> explain plan for
  2  select c.first, c.last, i.itemdesc
  3  from customer c, item i, inventory inv, cust_order co, orderline ol
  4  where c.custid = co.custid AND
  5  co.orderid = ol.orderid AND
  6  ol.invid = inv.invid AND
  7  inv.itemid = i.itemid AND
  8  c.first NOT IN (select fname from student);

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                              COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                          9
         1 FILTER
         2 HASH JOIN                                 9
         3 HASH JOIN                                 7
         4 HASH JOIN                                 5
         5 HASH JOIN                                 3
         6 TABLE ACCESS                              1
         7 TABLE ACCESS                              1
         8 TABLE ACCESS                              1
         9 TABLE ACCESS                              1
        10 TABLE ACCESS                              1
        11 TABLE ACCESS                              1

12 rows selected.
```

The above illustrates the cost in the NOT IN. It makes use of many HASH JOINs where the most costly operations were performed.

Nested Query with NOT IN operator with Parallelism

The tables were given degree of two (2).

```
SQL> ALTER TABLE customer parallel (degree 2);

Table altered.

SQL> ALTER TABLE cust_order parallel (degree 2);

Table altered.

SQL> ALTER TABLE orderline parallel (degree 2);

Table altered.
```

```
SQL> select /*+ parallel (student, 2) */
  2  c.first, c.last, i.itemdesc
  3  from customer c, item i, inventory inv, cust_order co, orderline ol
  4  where c.custid = co.custid AND
  5  co.orderid = ol.orderid AND
  6  ol.invid = inv.invid AND
  7  inv.itemid = i.itemid AND
  8  c.first NOT IN (select fname from student);

FIRST                LAST                 ITEMDESC
-------------------- -------------------- --------------------------------
Alissa               Chang                3-Season Tent
Alissa               Chang                Women's Hiking Shorts
Alissa               Chang                Women's Hiking Shorts
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P001                6        190
         1          0 Consumer   P000                2         74
         1          0 Producer   QC                  8        264
         1          1 Consumer   P001                3         81
         1          1 Consumer   P000                1         59
         1          1 Producer   P003                0         48
         1          1 Producer   P002                4         92
         1          2 Consumer   P003                3        124
         1          2 Consumer   P002                1         68
         1          2 Producer   P001                3        124
         1          2 Producer   P000                1         68

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          3 Consumer   P003                2         68
         1          3 Consumer   P002                4         88
         1          3 Producer   P001                0         48
         1          3 Producer   P000                6        108
         1          4 Consumer   P001                0         48
         1          4 Consumer   P000                4        144
         1          4 Producer   P003                3        124
         1          4 Producer   P002                1         68
         1          5 Consumer   P001                2         87
         1          5 Consumer   P000                4        126
         1          5 Producer   P003                0         48

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          5 Producer   P002                6        165
         1          6 Consumer   QC                  4        184
         1          6 Producer   P001                0         24
         1          6 Producer   P000                4        160

26 rows selected.
```

```
SQL> explain plan for
  2  select c.first, c.last, i.itemdesc
  3  from customer c, item i, inventory inv, cust_order co, orderline ol
  4  where c.custid = co.custid AND
  5  co.orderid = ol.orderid AND
  6  ol.invid = inv.invid AND
  7  inv.itemid = i.itemid AND
  8  c.first NOT IN (select fname from student);

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                           COST
---------- ---------------------------- ----------
         0 SELECT STATEMENT                       9
         1 FILTER
         2 HASH JOIN                              9
         3 HASH JOIN                              7
         4 HASH JOIN                              5
         5 HASH JOIN                              3
         6 TABLE ACCESS                           1
         7 TABLE ACCESS                           1
         8 TABLE ACCESS                           1
         9 TABLE ACCESS                           1
        10 TABLE ACCESS                           1

        ID OPERATION                           COST
---------- ---------------------------- ----------
        11 TABLE ACCESS                           1

12 rows selected.
```

Here we can see that HASH JOIN is quite costly operation to do.

5.9 ANY

Nested Query with ANY operator

The ANY operator is true if any value returned meets the condition.

The tables are given parallel degree of one (1), which is running in serial.

```
SQL> ALTER TABLE customer parallel (degree 1);

Table altered.

SQL> alter table cust_order parallel (degree 1);

Table altered.

SQL> alter table orderline parallel (degree 1);

Table altered.
```

```
SQL> select distinct c.first, co.methpmt, o.quantity
  2  from customer c, cust_order co, orderline o
  3  where c.custid = co.custid AND
  4  co.orderid = o.orderid AND
  5  o.quantity > ANY (select avg(quantity) from orderline);

FIRST                METHPMT     QUANTITY
-------------------- --------- ----------
Alissa               CC                 3
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

```
SQL> explain plan for
  2  select distinct c.first, co.methpmt, o.quantity
  3  from customer c, cust_order co, orderline o
  4  where c.custid = co.custid AND
  5  co.orderid = o.orderid AND
  6  o.quantity > ANY (select avg(quantity) from orderline);

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                        7
         1 SORT                                    7
         2 FILTER
         3 HASH JOIN                               5
         4 HASH JOIN                               3
         5 TABLE ACCESS                            1
         6 TABLE ACCESS                            1
         7 TABLE ACCESS                            1
         8 SORT
         9 TABLE ACCESS                            1

10 rows selected.
```

```
SQL> select sum(cost) as "Total Cost"
  2  from plan_table;

Total Cost
----------
        26
```

By observing, the above query has a total cost of 26. It is here we can see that FILTER and SORT doesn't really have a cost on top of all the other operations to satisfy this query.

Nested Query with ANY operator and parallelism

```
SQL> ALTER TABLE customer parallel (degree 2);

Table altered.

SQL> ALTER TABLE cust_order parallel (degree 2);

Table altered.

SQL> ALTER TABLE orderline parallel (degree 2);

Table altered.
```

```
SQL> select distinct c.first, co.methpmt, o.quantity
  2  from customer c, cust_order co, orderline o
  3  where c.custid = co.custid AND
  4  co.orderid = o.orderid AND
  5  o.quantity > ANY (select avg(quantity) from orderline);

FIRST                METHPMT     QUANTITY
-------------------- --------- ----------
Alissa               CC                 3
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P003                2         71
         1          0 Consumer   P002                4         95
         1          0 Producer   P001                0         48
         1          0 Producer   P000                6        118
         1          1 Consumer   P003                1         62
         1          1 Consumer   P002                5        121
         1          1 Producer   P001                0         48
         1          1 Producer   P000                6        135
         1          2 Consumer   P001                2         82
         1          2 Consumer   P000                4        113
         1          2 Producer   P003                1         64

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          2 Producer   P002                5        131
         1          3 Consumer   P001                3         75
         1          3 Consumer   P000                1         57
         1          3 Producer   P003                0         48
         1          3 Producer   P002                4         84
         1          4 Producer   P001                3         72
         1          4 Producer   P000                1         39

18 rows selected.
```

```
SQL> EXPLAIN PLAN FOR
  2  select distinct c.first, co.methpmt, o.quantity
  3  from customer c, cust_order co, orderline o
  4  where c.custid = co.custid AND
  5  co.orderid = o.orderid AND
  6  o.quantity > ANY (select avg(quantity) from orderline);

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                        7
         1 SORT                                    7
         2 FILTER
         3 HASH JOIN                               5
         4 HASH JOIN                               3
         5 TABLE ACCESS                            1
         6 TABLE ACCESS                            1
         7 TABLE ACCESS                            1
         8 SORT
         9 SORT
        10 TABLE ACCESS                            1

11 rows selected.
```

```
SQL> select sum(cost) as "Total Cost"
  2  from plan_table;

Total Cost
----------
        26
```

Again, with three tables being accessed, and a parallel hint, each table uses one process each to process their part. This query has a cost of 26, same as the non-parallel one.

5.10 EXIST

Nested Query with EXIST operator

The tables are given parallel degree of one (1), which is running in serial.

```
SQL> ALTER TABLE customer parallel (degree 1);

Table altered.

SQL> alter table cust_order parallel (degree 1);

Table altered.

SQL> alter table orderline parallel (degree 1);

Table altered.
```

```
SQL> select c.first, c.last, i.itemdesc
  2  from customer c, item i, inventory inv, cust_order co, orderline ol
  3  where c.custid = co.custid AND
  4  co.orderid = ol.orderid AND
  5  ol.invid = inv.invid AND
  6  inv.itemid = i.itemid AND
  7  exists (select * from student
  8  where fname = 'pete');

FIRST                LAST
-------------------- --------------------
ITEMDESC
--------------------------------------------------
Alissa               Chang
Women's Hiking Shorts

Alissa               Chang
Women's Hiking Shorts

Alissa               Chang
3-Season Tent


FIRST                LAST
-------------------- --------------------
ITEMDESC
--------------------------------------------------
Paula                Harris
3-Season Tent
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

There is no use of parallelism here, thus there are no rows selected or recorded in the parallel statistics record table.

```
SQL> explain plan for
  2  select c.first, c.last, i.itemdesc
  3  from customer c, item i, inventory inv, cust_order co, orderline ol
  4  where c.custid = co.custid AND
  5  co.orderid = ol.orderid AND
  6  ol.invid = inv.invid AND
  7  inv.itemid = i.itemid AND
  8  exists (select * from student
  9  where fname = 'pete');

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ---------------------------- ----------
         0 SELECT STATEMENT                        9
         1 FILTER
         2 HASH JOIN                               9
         3 HASH JOIN                               7
         4 HASH JOIN                               5
         5 HASH JOIN                               3
         6 TABLE ACCESS                            1
         7 TABLE ACCESS                            1
         8 TABLE ACCESS                            1
         9 TABLE ACCESS                            1
        10 TABLE ACCESS                            1

        ID OPERATION                            COST
---------- ---------------------------- ----------
        11 TABLE ACCESS                            1

12 rows selected.
```

Here, we see that since there are JOINS, the operations will be very costly. By observing the other operations, they merely have one (1) cost which is minimal compared to the JOIN statements.

Nested Query with EXIST operator, with parallelism

The tables were given all degree of two (2).

```
SQL> ALTER TABLE customer parallel (degree 2);

Table altered.

SQL> ALTER TABLE cust_order parallel (degree 2);

Table altered.

SQL> ALTER TABLE orderline parallel (degree 2);

Table altered.
```

```
SQL> select c.first, c.last, i.itemdesc
  2  from customer c, item i, inventory inv, cust_order co, orderline ol
  3  where c.custid = co.custid AND
  4  co.orderid = ol.orderid AND
  5  ol.invid = inv.invid AND
  6  inv.itemid = i.itemid AND
  7  exists (select * from student
  8  where fname = 'pete');

FIRST                LAST
-------------------- --------------------
ITEMDESC
--------------------------------------------------
Alissa               Chang
Women's Hiking Shorts

Alissa               Chang
Women's Hiking Shorts

Alissa               Chang
3-Season Tent


FIRST                LAST
-------------------- --------------------
ITEMDESC
--------------------------------------------------
Paula                Harris
3-Season Tent
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P002                2         46
         1          0 Consumer   P003                6         90
         1          0 Producer   QC                  8        136
         1          1 Consumer   P000               10        200
         1          1 Consumer   P001               20        359
         1          1 Producer   QC                 30        559
         1          2 Consumer   P002                4        126
         1          2 Consumer   P003                2         87
         1          2 Producer   P000                6        165
         1          2 Producer   P001                0         48
         1          3 Consumer   P002                5         98
         1          3 Consumer   P003                1         58
         1          3 Producer   P000                6        108
         1          3 Producer   P001                0         48
         1          4 Consumer   P000                4        127
         1          4 Consumer   P001                2         88
         1          4 Producer   P002                5        146
         1          4 Producer   P003                1         69
         1          5 Consumer   P000                1         59
         1          5 Consumer   P001                3         81
         1          5 Producer   P002                4         92
         1          5 Producer   P003                0         48
         1          6 Consumer   P002                1         69
         1          6 Consumer   P003                3        111
         1          6 Producer   P000                1         69
         1          6 Producer   P001                3        111
         1          7 Consumer   P000                4        128
         1          7 Consumer   P001                0         48
         1          7 Producer   P002                1         68
         1          7 Producer   P003                3        108
         1          8 Consumer   QC                  4        184
         1          8 Producer   P000                4        160
         1          8 Producer   P001                0         24
```

This operation have used many producers and consumers swapping information amongst each other. This is the parallelism working.

Lets see what the plan will show.

```
SQL> explain plan for
  2  select c.first, c.last, i.itemdesc
  3  from customer c, item i, inventory inv, cust_order co, orderline ol
  4  where c.custid = co.custid AND
  5  co.orderid = ol.orderid AND
  6  ol.invid = inv.invid AND
  7  inv.itemid = i.itemid AND
  8  exists (select * from student
  9  where fname = 'pete');

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                              COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                          9
         1 FILTER
         2 HASH JOIN                                 9
         3 HASH JOIN                                 7
         4 HASH JOIN                                 5
         5 HASH JOIN                                 3
         6 TABLE ACCESS                              1
         7 TABLE ACCESS                              1
         8 TABLE ACCESS                              1
         9 TABLE ACCESS                              1
        10 TABLE ACCESS                              1
        11 TABLE ACCESS                              1

12 rows selected.
```

It shows that the most work is done during the HASH JOINS. Thus we need to be very careful when joining tables because it is very costly operation.

CHAPTER 6 - GROUP BY

The GROUP BY clause is used to group selected rows and return a single row of summary information. Oracle collects each group of rows based on the values of the expression(s) specified in the GROUP BY clause.
If a SELECT statement contains the GROUP BY clause, the select list can contain only the following types of expressions:

- Constants
- Group functions
- The functions USER, UID, and SYSDATE
- Expressions identical to those in the GROUP BY clause
- Expressions involving the above expressions that evaluate to the same value for all rows in a group

Expressions in the GROUP BY clause can contain any columns in the tables, views, and snapshots in the FROM clause, regardless of whether the columns appear in the select list.

The GROUP BY clause can contain no more than 255 expressions. The total number of bytes in all expressions in the GROUP BY clause is limited to the size of a data block minus some overhead.

The following is a list of the queries that was experimented:
SINGLE TABLE:

- Normal Group BY
- Normal Group BY with parallelism
- Grouping by more than one column
- Grouping by more than one column with parallelism
- Grouping with Where clause
- Grouping with Where clause and parallelism
- Grouping with more than one columns and more than one Where clauses
- Grouping with more than one columns and more than one Where clauses with parallelism.
- Group by on primary key
- Group by on primary key, with parallelism
- Simple Group by with the HAVING clause
- Simple Group by with the HAVING clause and with parallelism
- Simple Group by with the HAVING clause on more than one column
- Simple Group by with the HAVING clause on more than one column, with parallelism
- Simple Group by with the HAVING clause and WHERE clause on more than one column.
- Simple Group by with the HAVING clause and WHERE clause on more than one column with parallelism.
- Grouping with HAVING, with more than one columns and more than one Where clauses
- Grouping with HAVING, with more than one columns and more than one Where clauses, with parallelism.
- Group by with HAVING on primary key
- Group by with HAVING on primary key, with parallelism

MULTIPLE TABLES:

- Normal Group BY with multiple tables, without join and with parallelism of 2 for each table.
- Normal Group BY with multiple tables
- Normal Group BY with multiple tables and parallelism.
- Group BY with multiple tables on Primary Key
- Group BY with multiple tables on Primary Key with parallelism
- Simple Group by with the HAVING clause on multiple tables
- Simple Group by with the HAVING clause on multiple tables with parallelism
- Group by not the same as Join attrib with parallelism

6.1 GROUP BY

Normal Group BY

```
SQL> select fname, count(*)
  2  from student
  3  group by fname;

FNAME                  COUNT(*)
-------------------- ----------
Paula                         1
bob                           1
craig                         1
dawn                          1
jack                          2
james                         1
john                          1
jones                         1
mel                           1
pat                           1
paul                          2
pete                          1
rick                          1
sam                           1
stan                          2
tim                           1
tom                           1
wang                          1

18 rows selected.
```

The above returns the first names of the students and the number of times the first names that the students occur.

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

There is no use of parallelism here, thus there are no rows selected or recorded in the parallel statistics record table.

```
SQL> explain plan for
  2  select fname, count(*)
  3  from student
  4  group by fname;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                           COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT
         1 SORT
         2 TABLE ACCESS
```

For this query, there were minimal accesses to the tables or operations. The only operation that was additional was the SORT, which was needed for the group by operation.

If the same GROUP BY clause was used with parallelism, there may be a difference in the overall performance. Lets have a look,

Normal Group BY, with parallelism

```
SQL> select /*+ parallel (student, 3) */
  2  fname, count(*)
  3  from student
  4  group by fname;

FNAME                  COUNT(*)
-------------------- ----------
bob                           1
craig                         1
jack                          2
john                          1
paul                          2
rick                          1
sam                           1
tom                           1
wang                          1
Paula                         1
dawn                          1
james                         1
jones                         1
mel                           1
pat                           1
pete                          1
stan                          2
tim                           1

18 rows selected.
```

From here, we can already see a difference in the order of the results reported. So there is some kind of difference between the serial query and the parallel query.

Doing further analysis on this we yield the following:

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P001                9        199
         1          0 Consumer   P000                9        201
         1          0 Producer   P003               18        352
         1          0 Producer   P004                0         48
         1          1 Consumer   QC                 18        226
         1          1 Producer   P001                9        112
         1          1 Producer   P000                9        114

7 rows selected.
```

```
SQL> explain plan for
  2  select /*+ parallel (student, 3) */
  3  fname, count(*)
  4  from student
  5  group by fname;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                        3
         1 SORT                                    3
         2 SORT                                    3
         3 TABLE ACCESS                            1
```

From this, we can see that it uses two (2) sorts rather than on, and the costs are doubled due to the two sorts.

Grouping by more than one column

```
SQL> select fname, lname, hcolor, avg(age)
  2  from student
  3  group by fname, hcolor, lname;

FNAME                LNAME                HCOLOR       AVG(AGE)
-------------------- -------------------- ---------- ----------
Paula                Marsh                Brown              16
bob                  smart                black              15
craig                stone                brown              34
dawn                 mal                  brown              66
jack                 jones                blond              55
jack                 brown                none               26
james                long                 brown              15
john                 black                black              44
jones                Ng                   black              23
mel                  hack                 green              12
pat                  stone                brown              45
paul                 jones                black              22
paul                 craz                 grey               77
pete                 line                 Black              19
rick                 sam                  brown              25
sam                  taps                 blond              21
stan                 short                blond              55
stan                 jack                 white              33
tim                  cray                 black              55
tom                  spat                 red                66
wang                 chris                grey               21

21 rows selected.
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

There is no use of parallelism here, thus there are no rows selected or recorded in the parallel statistics record table.

```
SQL> explain plan for
  2  select fname, lname, hcolor, avg(age)
  3  from student
  4  group by fname, hcolor, lname;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ---------------------------- ----------
         0 SELECT STATEMENT
         1 SORT
         2 TABLE ACCESS
```

We can see here, that although there were more group by columns, there is still the same amount of SORT statements and operations for the serial part. Lets have a look at the parallel hinted version.

Grouping by more than one column with parallelism

```
SQL> select /*+ parallel (student, 3) */
  2  fname, lname, hcolor, avg(age)
  3  from student
  4  group by fname, hcolor, lname;

FNAME                LNAME                HCOLOR       AVG(AGE)
-------------------- -------------------- ---------- ----------
Paula                Marsh                Brown              16
bob                  smart                black              15
craig                stone                brown              34
dawn                 mal                  brown              66
jack                 jones                blond              55
jack                 brown                none               26
james                long                 brown              15
jones                Ng                   black              23
mel                  hack                 green              12
pat                  stone                brown              45
paul                 jones                black              22
paul                 craz                 grey               77
rick                 sam                  brown              25
tom                  spat                 red                66
wang                 chris                grey               21
john                 black                black              44
pete                 line                 Black              19
sam                  taps                 blond              21
stan                 short                blond              55
stan                 jack                 white              33
tim                  cray                 black              55

21 rows selected.
```

Just like before, we can see that the results were reportedly different to that of the serial version.

Lets do further investigation:

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P001               15        842
         1          0 Consumer   P000                6        366
         1          0 Producer   P004                0         48
         1          0 Producer   P003               21       1160
         1          1 Consumer   QC                 21        530
         1          1 Producer   P001               15        368
         1          1 Producer   P000                6        162

7 rows selected.
```

```
SQL> explain plan for
  2  select /*+ parallel (student, 3) */
  3  fname, lname, hcolor, avg(age)
  4  from student
  5  group by fname, hcolor, lname;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                              COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                        3
         1 SORT                                    3
         2 SORT                                    3
         3 TABLE ACCESS                            1
```

Here we can see that there are two sorts compared to one. This is quite similar to the previous singular grouping by just one column.

6.2 WHERE

Grouping with Where clause

Here we specify a more descriptive clause using the Where clause. For demonstration purposes only I have used two different student names and did an average on the two. This is jus to show that grouping by clause requires all specified to view as result to be in the group by clause.

```
SQL> select fname, avg(age), count(*)
  2  from student
  3  where age > 16
  4  group by fname;

FNAME                 AVG(AGE)   COUNT(*)
-------------------- ---------- ----------
craig                       34          1
dawn                        66          1
jack                      40.5          2
john                        44          1
jones                       23          1
pat                         45          1
paul                      49.5          2
pete                        19          1
rick                        25          1
sam                         21          1
stan                        44          2

FNAME                 AVG(AGE)   COUNT(*)
-------------------- ---------- ----------
tim                         55          1
tom                         66          1
wang                        21          1

14 rows selected.
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

There are no uses of parallelism here, thus there are no rows selected or recorded in the parallel statistics record table.

```
SQL> explain plan for
  2  select fname, avg(age), count(*)
  3  from student
  4  where age > 16
  5  group by fname;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                              COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT
         1 SORT
         2 TABLE ACCESS
```

By viewing the above, it is quite similar to the previous select statements with group by clauses in that they all are using the same amount of operations to satisfy the query.

Lets see if there are any major differences to that of the parallel hinted version:

Grouping with Where clause and parallelism

The following is hinted with degree of two (2) for the purpose of demonstrating parallelism and serial execution.

```
SQL> select /*+ parallel (student, 2) */
  2  fname, avg(age), count(*)
  3  from student
  4  where age > 16
  5  group by fname;

FNAME                  AVG(AGE)   COUNT(*)
-------------------- ---------- ----------
craig                        34          1
jack                       40.5          2
john                         44          1
paul                       49.5          2
rick                         25          1
sam                          21          1
tom                          66          1
wang                         21          1
dawn                         66          1
jones                        23          1
pat                          45          1

FNAME                  AVG(AGE)   COUNT(*)
-------------------- ---------- ----------
pete                         19          1
stan                         44          2
tim                          55          1

14 rows selected.
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P001                8        431
         1          0 Consumer   P000                6        335
         1          0 Producer   P003                0         48
         1          0 Producer   P002               14        718
         1          1 Consumer   QC                 14        244
         1          1 Producer   P001                8        137
         1          1 Producer   P000                6        107

7 rows selected.
```

```
SQL> explain plan for
  2  select /*+ parallel (student, 2) */
  3  fname, avg(age), count(*)
  4  from student
  5  where age > 16
  6  group by fname;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                        3
         1 SORT                                    3
         2 SORT                                    3
         3 TABLE ACCESS                            1
```

As we can observe here, there is no difference when executed in serial throughout the three examples that was performed. There is also no difference in the operation counts or the cost differences between the three different parallel-executed queries.

Lets have a few more experiments to see whether additional operations will add to the cost and processes used.

Grouping with more than one columns and more than one Where clauses

```
SQL>  select fname, lname, avg(age) as Aage, count(*)
  2   from student
  3   where age > 15 AND
  4  fname = 'pete' OR
  5  fname = 'john'
  6  group by fname, lname;

FNAME                LNAME                      AAGE   COUNT(*)
-------------------- -------------------- ---------- ----------
john                 black                        44          1
pete                 line                         19          1
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

There are no uses of parallelism here, thus there are no rows selected or recorded in the parallel statistics record table.

```
SQL> explain plan for
  2  select fname, lname, avg(age) as Aage, count(*)
  3  from student
  4  where age > 15 AND
  5  fname = 'pete' OR
  6  fname = 'john'
  7  group by fname, lname;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT
         1 SORT
         2 TABLE ACCESS
```

The above still looks the same as the previous experiments, so lets have a look at the parallel version.

Grouping with more than one columns and more than one Where clauses with parallelism.

```
SQL> select /*+ parallel (student, 2) */
  2  fname, lname, avg(age) as Aage, count(*)
  3  from student
  4  where age > 15 AND
  5  fname = 'pete' OR
  6  fname = 'john'
  7  group by fname, lname;

FNAME                LNAME                      AAGE   COUNT(*)
-------------------- -------------------- ---------- ----------
john                 black                        44          1
pete                 line                         19          1
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P001                2        157
         1          0 Consumer   P000                0         48
         1          0 Producer   P002                2        157
         1          0 Producer   P003                0         48
         1          1 Consumer   QC                  2         89
         1          1 Producer   P001                2         65
         1          1 Producer   P000                0         24
```

```
SQL> explain plan for
  2  select /*+ parallel (student, 2) */
  3  fname, lname, avg(age) as Aage, count(*)
  4  from student
  5  where age > 15 AND
  6  fname = 'pete' OR
  7  fname = 'john'
  8  group by fname, lname;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                        3
         1 SORT                                    3
         2 SORT                                    3
         3 TABLE ACCESS                            1
```

No differences were found.

6.3 PRIMARY KEY

Group by on primary key

```
SQL> select sid, fname, count(*), avg(age)
  2  from student
  3  group by sid, fname;

       SID FNAME                  COUNT(*)   AVG(AGE)
---------- -------------------- ---------- ----------
         1 bob                           1         15
         2 sam                           1         21
         3 craig                         1         34
         4 tom                           1         66
         5 pat                           1         45
         6 tim                           1         55
         7 paul                          1         77
         8 dawn                          1         66
         9 jack                          1         55
        10 mel                           1         12
        11 jones                         1         23

       SID FNAME                  COUNT(*)   AVG(AGE)
---------- -------------------- ---------- ----------
        12 stan                          1         33
        14 rick                          1         25
        15 paul                          1         22
        16 james                         1         15
        17 jack                          1         26
        18 stan                          1         55
        19 wang                          1         21
        20 john                          1         44
       989 pete                          1         19
       999 Paula                         1         16

21 rows selected.
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

There are no uses of parallelism here, thus there are no rows selected or recorded in the parallel statistics record table.

```
SQL> explain plan for
  2  select sid, fname, count(*), avg(age)
  3  from student
  4  group by sid, fname;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT
         1 SORT
         2 TABLE ACCESS
```

This is something that was not really expected. To see all processes using the same amount of operations and cost for serial queries and then all the processes the same with the parallel queries are very surprising.

Group by on primary key, with parallelism

```
SQL> select /*+ parallel (student, 2) */
  2  sid, count(*), avg(age)
  3  from student
  4  group by sid;

       SID   COUNT(*)   AVG(AGE)
---------- ---------- ----------
         1          1         15
         3          1         34
         4          1         66
         7          1         77
         9          1         55
        10          1         12
        12          1         33
        14          1         25
        15          1         22
        16          1         15
        17          1         26

       SID   COUNT(*)   AVG(AGE)
---------- ---------- ----------
        19          1         21
       999          1         16
         2          1         21
         5          1         45
         6          1         55
         8          1         66
        11          1         23
        18          1         55
        20          1         44
       989          1         19

21 rows selected.
```

At this point there is one difference that can be seen her. The order of the reported result differs the serial version.

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P001               13        647
         1          0 Consumer   P000                8        417
         1          0 Producer   P003                0         48
         1          0 Producer   P002               21       1016
         1          1 Consumer   QC                 21        302
         1          1 Producer   P001               13        181
         1          1 Producer   P000                8        121
```

```
SQL> explain plan for
  2  select sid, count(*), avg(age)
  3  from student
  4  group by sid;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                           COST
---------- ----------------------------- ----------
         0 SELECT STATEMENT
         1 SORT
         2 TABLE ACCESS

3 rows selected.
```

Here we see something different. The parallel hinted query for the primary key can clearly be seen here to use the same amount of operations as the serial.

6.4 HAVING

Having Clause

The HAVING clause is used to restrict which groups of rows defined by the GROUP BY clause is returned by the query. Oracle processes the WHERE, GROUP BY, and HAVING clauses in the following manner:

- If the statement contains a WHERE clause, Oracle eliminates all rows that do not satisfy it.
- Oracle calculates and forms the groups as specified in the GROUP BY clause.
- Oracle removes all groups that do not satisfy the HAVING clause.

Specify the GROUP BY and HAVING clauses after the WHERE and CONNECT BY clauses. If both the GROUP BY and HAVING clauses are specified, they can appear in either order.

Simple Group by with the HAVING clause

```
SQL> select fname, count(*), sum(age)
  2  from student
  3  group by fname
  4  having sum(age) > 16;

FNAME                  COUNT(*)   SUM(AGE)
-------------------- ---------- ----------
craig                         1         34
dawn                          1         66
jack                          2         81
john                          1         44
jones                         1         23
pat                           1         45
paul                          2         99
pete                          1         19
rick                          1         25
sam                           1         21
stan                          2         88

FNAME                  COUNT(*)   SUM(AGE)
-------------------- ---------- ----------
tim                           1         55
tom                           1         66
wang                          1         21

14 rows selected.
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;
```

```
no rows selected
```

There are no uses of parallelism here, thus there are no rows selected or recorded in the parallel statistics record table.

```
SQL> explain plan for
  2  select fname, count(*), sum(age)
  3  from student
  4  group by fname
  5  having sum(age) > 16;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                           COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT
         1 SORT
         2 TABLE ACCESS
```

View the results above, it shows not that much difference with the other queries where the SORT is the major operation that group by uses.

Simple Group by with the HAVING clause and with parallelism

```
SQL> select /*+ parallel (student, 3) */
  2  fname, count(*), sum(age)
  3  from student
  4  group by fname
  5  having sum(age) > 16;

FNAME                  COUNT(*)   SUM(AGE)
-------------------- ---------- ----------
dawn                          1         66
jones                         1         23
pat                           1         45
pete                          1         19
stan                          2         88
tim                           1         55
craig                         1         34
jack                          2         81
john                          1         44
paul                          2         99
rick                          1         25

FNAME                  COUNT(*)   SUM(AGE)
-------------------- ---------- ----------
sam                           1         21
tom                           1         66
wang                          1         21

14 rows selected.
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P001                9        406
         1          0 Consumer   P000                9        408
         1          0 Producer   P004                0         48
         1          0 Producer   P003               18        766
         1          1 Consumer   QC                 14        242
         1          1 Producer   P001                8        135
         1          1 Producer   P000                6        107
```

```
SQL> explain plan for
  2  select /*+ parallel (student, 3) */
  3  fname, count(*), sum(age)
  4  from student
  5  group by fname
  6  having sum(age) > 16;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                        3
         1 FILTER
         2 SORT                                    3
         3 SORT                                    3
         4 TABLE ACCESS                            1
```

From the results above, it can be seen that the HAVING clause uses a filter which determines the selection of the output. It also, adds an extra sorting algorithm that adds to the overhead for this query.

Simple Group by with the HAVING clause on more than one column

```
SQL> select fname, lname, hcolor, avg(age)
  2  from student
  3  group by fname, hcolor, lname
  4  having sum(age) > 16;

FNAME                LNAME                HCOLOR       AVG(AGE)
-------------------- -------------------- ---------- ----------
craig                stone                brown              34
dawn                 mal                  brown              66
jack                 jones                blond              55
jack                 brown                none               26
john                 black                black              44
jones                Ng                   black              23
pat                  stone                brown              45
paul                 jones                black              22
paul                 craz                 grey               77
pete                 line                 Black              19
rick                 sam                  brown              25

FNAME                LNAME                HCOLOR       AVG(AGE)
-------------------- -------------------- ---------- ----------
sam                  taps                 blond              21
stan                 short                blond              55
stan                 jack                 white              33
tim                  cray                 black              55
tom                  spat                 red                66
wang                 chris                grey               21

17 rows selected.
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

There are no uses of parallelism here, thus there are no rows selected or recorded in the parallel statistics record table.

```
SQL> explain plan for
  2  select fname, lname, hcolor, avg(age)
  3  from student
  4  group by fname, hcolor, lname
  5  having sum(age) > 16;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT
         1 FILTER
         2 SORT
         3 TABLE ACCESS
```

We can start to see a pattern here as the HAVING clause merely adds a filter to the operations where there GROUP BY clause does a sorting algorithm to the table.

Simple Group by with the HAVING clause on more than one column, with parallelism

```
SQL> select /*+ parallel (student, 4) */
  2  fname, lname, hcolor, avg(age)
  3  from student
  4  group by fname, hcolor, lname
  5  having sum(age) > 16;

FNAME                LNAME                HCOLOR       AVG(AGE)
-------------------- -------------------- ---------- ----------
john                 black                black              44
pete                 line                 Black              19
sam                  taps                 blond              21
stan                 short                blond              55
stan                 jack                 white              33
tim                  cray                 black              55
craig                stone                brown              34
dawn                 mal                  brown              66
jack                 jones                blond              55
jack                 brown                none               26
jones                Ng                   black              23

FNAME                LNAME                HCOLOR       AVG(AGE)
-------------------- -------------------- ---------- ----------
pat                  stone                brown              45
paul                 jones                black              22
paul                 craz                 grey               77
rick                 sam                  brown              25
tom                  spat                 red                66
wang                 chris                grey               21

17 rows selected.
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P001               15       1187
         1          0 Consumer   P000                6        504
         1          0 Producer   P004               21       1643
         1          0 Producer   P002                0         48
         1          1 Consumer   QC                 17        504
         1          1 Producer   P001               11        318
         1          1 Producer   P000                6        186

7 rows selected.
```

```
SQL> explain plan for
  2  select /*+ parallel (student, 4) */
  3  fname, lname, hcolor, avg(age)
  4  from student
  5  group by fname, hcolor, lname
  6  having sum(age) > 16;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ----------------------------- ----------
         0 SELECT STATEMENT                        4
         1 FILTER
         2 SORT                                    4
         3 SORT                                    4
         4 TABLE ACCESS                            1
```

Here we see that with parallelism added onto this query, there is an extra SORT.

6.5 WHERE

Simple Group by with the HAVING clause and WHERE clause on more than one column.

```
SQL> select fname, hcolor, avg(age), count(*)
  2  from student
  3  where hcolor = 'black'
  4  group by fname, hcolor
  5  having sum(age) > 16;

FNAME                HCOLOR       AVG(AGE)   COUNT(*)
-------------------- ---------- ---------- ----------
john                 black              44          1
jones                black              23          1
paul                 black              22          1
tim                  black              55          1
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

There are no uses of parallelism here, thus there are no rows selected or recorded in the parallel statistics record table.

```
SQL> explain plan for
  2  select fname, hcolor, avg(age), count(*)
  3  from student
  4  where hcolor = 'black'
  5  group by fname, hcolor
  6  having sum(age) > 16;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ----------------------------- ----------
         0 SELECT STATEMENT
         1 FILTER
         2 SORT
         3 TABLE ACCESS
```

Adding a WHERE clause does not seem to add any more operations to the query.

Simple Group by with the HAVING clause and WHERE clause on more than one column with parallelism.

```
SQL> select /*+ parallel (student, 4) */
  2  fname, hcolor, avg(age), count(*)
  3  from student
  4  where hcolor = 'black'
  5  group by fname, hcolor
  6  having sum(age) > 16;

FNAME                HCOLOR       AVG(AGE)   COUNT(*)
-------------------- ---------- ---------- ----------
john                 black              44          1
paul                 black              22          1
jones                black              23          1
tim                  black              55          1
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P001                2        204
         1          0 Consumer   P000                3        281
         1          0 Producer   P004                5        437
         1          0 Producer   P002                0         48
         1          1 Consumer   QC                  4        148
         1          1 Producer   P001                2         74
         1          1 Producer   P000                2         74
```

Here we can see that the producers are here performing the SORTs where the consumers are the filters and select statements.

```
SQL> explain plan for
  2  select /*+ parallel (student, 4) */
  3  fname, hcolor, avg(age), count(*)
  4  from student
  5  where hcolor = 'black'
  6  group by fname, hcolor
  7  having sum(age) > 16;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ----------------------------- ----------
         0 SELECT STATEMENT                       3
         1 FILTER
         2 SORT                                   3
         3 SORT                                   3
         4 TABLE ACCESS                           1
```

An extra SORT was added as an operation just like the other examples where parallelism was used.

Grouping with HAVING, with more than one columns and more than one Where clauses

Here, more than one columns are selected and more than one WHERE clauses are used in order to see whether they all contribute to any changes. Together with the HAVING clause and the GROUP By clause, the following result was recorded.

```
SQL> select fname, lname, avg(age) as Aage, count(*)
  2  from student
  3  where age > 15 AND
  4  fname = 'pete' OR
  5  fname = 'john'
  6  group by fname, lname
  7  having count(*) < 5;

FNAME                LNAME                      AAGE   COUNT(*)
-------------------- -------------------- ---------- ----------
john                 black                        44          1
pete                 line                         19          1
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

Since there was no parallelism used, there are no rows selected in this case.

```
SQL> explain plan for
  2  select fname, lname, avg(age) as Aage, count(*)
  3  from student
  4  where age > 15 AND
  5  fname = 'pete' OR
  6  fname = 'john'
  7  group by fname, lname
  8  having count(*) < 5;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                             COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT
         1 FILTER
         2 SORT
         3 TABLE ACCESS
```

We can see here that there are not much changed from previous experiments. Lets continue to try other combinations.

Grouping with HAVING, with more than one columns and more than one Where clauses, with parallelism.

```
SQL> select /*+ parallel (student, 3) */
  2  fname, lname, avg(age) as Aage, count(*)
  3  from student
  4  where age > 15 AND
  5  fname = 'pete' OR
  6  fname = 'john'
  7  group by fname, lname
  8  having count(*) < 5;

FNAME                LNAME                      AAGE   COUNT(*)
-------------------- -------------------- ---------- ----------
john                 black                        44          1
pete                 line                         19          1
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P001                2        157
         1          0 Consumer   P000                0         48
         1          0 Producer   P004                0         48
         1          0 Producer   P003                2        157
         1          1 Consumer   QC                  2         89
         1          1 Producer   P001                2         65
         1          1 Producer   P000                0         24
```

```
SQL> explain plan for
  2  select /*+ parallel (student, 3) */
  3  fname, lname, avg(age) as Aage, count(*)
  4  from student
  5  where age > 15 AND
  6  fname = 'pete' OR
  7  fname = 'john'
  8  group by fname, lname
  9  having count(*) < 5;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                        3
         1 FILTER
         2 SORT                                    3
         3 SORT                                    3
         4 TABLE ACCESS                            1
```

With the use of parallelism, it can be seen that the changes are linear. Linear I mean by the fact that there are always that extra sort that parallelism has. The processes that are used here also change linearly.

Group by with HAVING on primary key

```
SQL> select sid, fname, count(*), avg(age)
  2  from student
  3  group by sid, fname
  4  having count(*) < 5;

       SID FNAME                  COUNT(*)   AVG(AGE)
---------- -------------------- ---------- ----------
         1 bob                           1         15
         2 sam                           1         21
         3 craig                         1         34
         4 tom                           1         66
         5 pat                           1         45
         6 tim                           1         55
         7 paul                          1         77
         8 dawn                          1         66
         9 jack                          1         55
        10 mel                           1         12
        11 jones                         1         23

       SID FNAME                  COUNT(*)   AVG(AGE)
---------- -------------------- ---------- ----------
        12 stan                          1         33
        14 rick                          1         25
        15 paul                          1         22
        16 james                         1         15
        17 jack                          1         26
        18 stan                          1         55
        19 wang                          1         21
        20 john                          1         44
       989 pete                          1         19
       999 Paula                         1         16

21 rows selected.
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

Since there was no parallelism used, there are no rows selected in this case.

```
SQL> explain plan for
  2  select sid, fname, count(*), avg(age)
  3  from student
  4  group by sid, fname
  5  having count(*) < 5;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT
         1 FILTER
         2 SORT
         3 TABLE ACCESS
```

Just like the other examples, there were little changes when extra clauses were used.

Group by with HAVING on primary key, with parallelism

```
SQL> select /*+ parallel (student, 4) */
  2  sid, fname, count(*), avg(age)
  3  from student
  4  group by sid, fname
  5  having count(*) < 5;

       SID FNAME                  COUNT(*)   AVG(AGE)
---------- -------------------- ---------- ----------
         1 bob                           1         15
         2 sam                           1         21
         3 craig                         1         34
         6 tim                           1         55
         9 jack                          1         55
        12 stan                          1         33
        15 paul                          1         22
        17 jack                          1         26
        18 stan                          1         55
        19 wang                          1         21
       989 pete                          1         19

       SID FNAME                  COUNT(*)   AVG(AGE)
---------- -------------------- ---------- ----------
       999 Paula                         1         16
         4 tom                           1         66
         5 pat                           1         45
         7 paul                          1         77
         8 dawn                          1         66
        10 mel                           1         12
        11 jones                         1         23
        14 rick                          1         25
        16 james                         1         15
        20 john                          1         44

21 rows selected.
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P001                9        515
         1          0 Consumer   P000               12        673
         1          0 Producer   P004               21       1140
         1          0 Producer   P002                0         48
         1          1 Consumer   QC                 21        426
         1          1 Producer   P001                9        185
         1          1 Producer   P000               12        241
```

```
SQL> explain plan for
  2  select /*+ parallel (student, 4) */
  3  sid, fname, count(*), avg(age)
  4  from student
  5  group by sid, fname
  6  having count(*) < 5;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                        3
         1 FILTER
         2 SORT                                    3
         3 SORT                                    3
         4 TABLE ACCESS                            1
```

There were no dramatic changes here either when the query was applied on the primary key.

Normal Group BY with multiple tables, without join and with parallelism of 2 for each table.

```
SQL> select c.first, cu.methpmt
  2  from customer c, cust_order cu
  3  group by c.first, cu.methpmt;

FIRST                METHPMT
-------------------- ---------
Alissa               CC
Mitch                CC
Mitch                CHECK
Paula                CC
Paula                CHECK
pete                 CC
pete                 CHECK
Alissa               CHECK
Lee                  CC
Lee                  CHECK
Maria                CC

FIRST                METHPMT
-------------------- ---------
Maria                CHECK

12 rows selected.
```

```
SQL> explain plan for
  2  select c.first, cu.methpmt
  3  from customer c, cust_order cu
  4  group by c.first, cu.methpmt;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                      188
         1 SORT                                  188
         2 MERGE JOIN                             83
         3 TABLE ACCESS                            1
         4 SORT                                  187
         5 TABLE ACCESS                            1

6 rows selected.
```

We can see here that sorting the two tables is a big task when there are no joins between the tables. There are duplicates here that were displayed unnecessarily, and thus a waste and not efficient. But again, the join statements require a fair amount of processing to complete the operation.

6.6 MULTIPLE TABLES

Normal Group BY with multiple tables

```
SQL> select c.first, cu.methpmt
  2  from customer c, cust_order cu
  3  where c.custid = cu.custid
  4  group by c.first, cu.methpmt;

FIRST                METHPMT
-------------------- ---------
Alissa               CC
Lee                  CC
Maria                CHECK
Mitch                CC
Paula                CC
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

There is no use of parallelism here, thus there are no rows selected or recorded in the parallel statistics record table.

```
SQL> explain plan for
  2  select c.first, cu.methpmt
  3  from customer c, cust_order cu
  4  where c.custid = cu.custid
  5  group by c.first, cu.methpmt;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                        5
         1 SORT                                    5
         2 HASH JOIN                               3
         3 TABLE ACCESS                            1
         4 TABLE ACCESS                            1
```

We can see here that two tables were accessed whilst one joining clause joins the two tables with a very expensive cost and then sorting the table before selecting the appropriate list of results.

Normal Group BY with multiple tables and parallelism.

```
SQL> select c.first, cu.methpmt
  2  from customer c, cust_order cu
  3  where c.custid = cu.custid
  4  group by c.first, cu.methpmt;

FIRST                METHPMT
-------------------- ---------
Lee                  CC
Maria                CHECK
Alissa               CC
Mitch                CC
Paula                CC
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P003                6         88
         1          0 Consumer   P002                6         88
         1          0 Producer   P001                0         48
         1          0 Producer   P000               12        128
         1          1 Consumer   P001               13        189
         1          1 Consumer   P000               23        309
         1          1 Producer   P003                0         48
         1          1 Producer   P002               36        450
         1          2 Consumer   QC                 12        194
         1          2 Producer   P001                5         85
         1          2 Producer   P000                7        109
```

```
SQL> explain plan for
  2  select c.first, cu.methpmt
  3  from customer c, cust_order cu
  4  where c.custid = cu.custid
  5  group by c.first, cu.methpmt;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                        5
         1 SORT                                    5
         2 HASH JOIN                               3
         3 TABLE ACCESS                            1
         4 TABLE ACCESS                            1
```

Here we can see that the cost of each operation has been reduced dramatically when we link the two tables appropriately. The HASH JOIN and SORT are still the most costly of the overall operation.

6.7 PRIMARY KEY

Group BY with multiple tables on Primary Key

```
SQL> select c.custid, c.first, cu.methpmt
  2  from customer c, cust_order cu
  3  where c.custid = cu.custid
  4  group by c.custid, c.first, cu.methpmt;

    CUSTID FIRST                METHPMT
---------- -------------------- ---------
       107 Paula                CC
       133 Maria                CHECK
       154 Lee                  CC
       179 Alissa               CC
       232 Mitch                CC
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

There is no use of parallelism here, thus there are no rows selected or recorded in the parallel statistics record table.

```
SQL> explain plan for
  2  select c.custid, c.first, cu.methpmt
  3  from customer c, cust_order cu
  4  where c.custid = cu.custid
  5  group by c.custid, c.first, cu.methpmt;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                              COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                          5
         1 SORT                                      5
         2 HASH JOIN                                 3
         3 TABLE ACCESS                              1
         4 TABLE ACCESS                              1
```

We can see here that are two table accesses whilst the JOIN still jumps with the cost. The sort and select statement is also very high with this query.

Group BY with multiple tables on Primary Key with parallelism

The tables are given a degree of two (2) for the purpose of demonstrating this query with parallelism.

```
SQL> ALTER TABLE customer parallel (degree 2);

Table altered.

SQL> ALTER TABLE cust_order parallel (degree 2);

Table altered.
```

```
SQL> select c.custid, c.first, cu.methpmt
  2  from customer c, cust_order cu
  3  where c.custid = cu.custid
  4  group by c.custid, c.first, cu.methpmt;

    CUSTID FIRST                METHPMT
---------- -------------------- ---------
       133 Maria                CHECK
       154 Lee                  CC
       179 Alissa               CC
       232 Mitch                CC
       107 Paula                CC
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P003                2         71
         1          0 Consumer   P002                4         95
         1          0 Producer   P001                0         48
         1          0 Producer   P000                6        118
         1          1 Consumer   P003                1         57
         1          1 Consumer   P002                5         96
         1          1 Producer   P001                0         48
         1          1 Producer   P000                6        105
         1          2 Consumer   P001                1         64
         1          2 Consumer   P000                5        131
         1          2 Producer   P003                1         64
         1          2 Producer   P002                5        131
         1          3 Consumer   QC                  5        130
         1          3 Producer   P001                1         40
         1          3 Producer   P000                4         90
```

```
SQL> explain plan for
  2  select c.custid, c.first, cu.methpmt
  3  from customer c, cust_order cu
  4  where c.custid = cu.custid
  5  group by c.custid, c.first, cu.methpmt;

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ----------------------------- ----------
         0 SELECT STATEMENT                        5
         1 SORT                                    5
         2 HASH JOIN                               3
         3 TABLE ACCESS                            1
         4 TABLE ACCESS                            1
```

Here the operations are the same as the serial one even though that we executed in parallel. The v$pq_tqstat table, however, tells us that it is running in parallel and different tasks are split in to different producers and consumers.

Simple Group by with the HAVING clause on multiple tables

The tables are changed back to serial execution.

```
SQL> ALTER TABLE customer parallel (degree 1);

Table altered.

SQL> ALTER TABLE cust_order parallel (degree 1);

Table altered.
```

The HAVING clause here filters down the list by the further check for “CC” from the cust_order table.

```
SQL> select c.custid, c.first, cu.methpmt
  2  from customer c, cust_order cu
  3  where c.custid = cu.custid
  4  group by c.custid, c.first, cu.methpmt
  5  having cu.methpmt = 'CC';

    CUSTID FIRST                METHPMT
---------- -------------------- ---------
       107 Paula                CC
       154 Lee                  CC
       179 Alissa               CC
       232 Mitch                CC
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

no rows selected
```

There is no use of parallelism here, thus there are no rows selected or recorded in the parallel statistics record table.

```
SQL> explain plan for
  2  select c.custid, c.first, cu.methpmt
  3  from customer c, cust_order cu
  4  where c.custid = cu.custid
  5  group by c.custid, c.first, cu.methpmt
  6  having cu.methpmt = 'CC';

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                              COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                          5
         1 FILTER
         2 SORT                                      5
         3 HASH JOIN                                 3
         4 TABLE ACCESS                              1
         5 TABLE ACCESS                              1

6 rows selected.
```

The operations from the plan table are the same as before.

Simple Group by with the HAVING clause on multiple tables with parallelism

The tables are changed to degree of two (2) for each of the used tables.

```
SQL> ALTER TABLE customer parallel (degree 2);

Table altered.

SQL> ALTER TABLE cust_order parallel (degree 2);

Table altered.
```

```
SQL>  select c.custid, c.first, cu.methpmt
  2   from customer c, cust_order cu
  3   where c.custid = cu.custid
  4   group by c.custid, c.first, cu.methpmt
  5   having cu.methpmt = 'CC';

    CUSTID FIRST                METHPMT
---------- -------------------- ---------
       107 Paula                CC
       154 Lee                  CC
       179 Alissa               CC
       232 Mitch                CC
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P003                2         71
         1          0 Consumer   P002                4         95
         1          0 Producer   P001                0         48
         1          0 Producer   P000                6        118
         1          1 Consumer   P003                1         57
         1          1 Consumer   P002                5         96
         1          1 Producer   P001                0         48
         1          1 Producer   P000                6        105
         1          2 Consumer   P001                1         64
         1          2 Consumer   P000                5        131
         1          2 Producer   P003                1         64
         1          2 Producer   P002                5        131
         1          3 Consumer   QC                  4        111
         1          3 Producer   P001                1         40
         1          3 Producer   P000                3         71
```

```
SQL> explain plan for
  2  select c.custid, c.first, cu.methpmt
  3  from customer c, cust_order cu
  4  where c.custid = cu.custid
  5  group by c.custid, c.first, cu.methpmt
  6  having cu.methpmt = 'CC';

Explained.
```

```
SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                        5
         1 FILTER
         2 SORT                                    5
         3 HASH JOIN                               3
         4 TABLE ACCESS                            1
         5 TABLE ACCESS                            1

6 rows selected.
```

From these results, it seems like there are very similar usage of the parallel system as they all pretty much use the same amount of processes.

Group by not the same as Join attrib with parallelism

Each table is given the parallelism of degree two (2).

```
SQL> ALTER TABLE customer parallel (degree 2);

Table altered.

SQL> ALTER TABLE cust_order parallel (degree 2);

Table altered.
```

```
SQL> select c.first, cu.methpmt
  2  from customer c, cust_order cu
  3  where c.custid = cu.custid
  4  group by c.last, c.first, cu.methpmt;

FIRST                METHPMT
-------------------- ---------
Alissa               CC
Mitch                CC
Paula                CC
Maria                CHECK
Lee                  CC
```

```
SQL> select dfo_number, tq_id, server_type, process, num_rows, bytes
  2  from v$pq_tqstat
  3  order by dfo_number, tq_id, server_type;

DFO_NUMBER      TQ_ID SERVER_TYP PROCESS      NUM_ROWS      BYTES
---------- ---------- ---------- ---------- ---------- ----------
         1          0 Consumer   P003                2         87
         1          0 Consumer   P002                4        126
         1          0 Producer   P001                0         48
         1          0 Producer   P000                6        165
         1          1 Consumer   P003                1         57
         1          1 Consumer   P002                5         96
         1          1 Producer   P001                0         48
         1          1 Producer   P000                6        105
         1          2 Consumer   P001                4        125
         1          2 Consumer   P000                2         87
         1          2 Producer   P003                1         68
         1          2 Producer   P002                5        144
         1          3 Consumer   QC                  5        145
         1          3 Producer   P001                3         82
         1          3 Producer   P000                2         63
```

```
SQL> explain plan for
  2  select c.first, cu.methpmt
  3  from customer c, cust_order cu
  4  where c.custid = cu.custid
  5  group by c.last, c.first, cu.methpmt;

Explained.

SQL> select id, operation, cost
  2  from plan_table;

        ID OPERATION                            COST
---------- ------------------------------ ----------
         0 SELECT STATEMENT                        5
         1 SORT                                    5
         2 HASH JOIN                               3
         3 TABLE ACCESS                            1
         4 TABLE ACCESS                            1
```

The above shows that having a different attrib from the group by and join statement does not make a very large difference to the operation at hand.

CHAPTER 7 – DML

Data manipulation language (DML) commands query and manipulate data in existing schema objects. The DML allows one to modify, update and delete certain records in tables with ease. Together with clauses such as WHERE, IN and other examples mentioned in the DLL section that can be used here, forms tool that is both flexible and easy to use.

7.1 PARALLEL DML

Parallel DML

When parallel DML is enabled for the session, all DML portions of statements issued are considered for parallel execution. Even with parallel DML enabled, some DML operations are restricted from using parallelised execution, while others may still execute serially unless parallel hints and clauses are specified.

The following restrictions apply to parallel DML operations:

- DML operations on clustered tables are not parallelised.
- DML operations with embedded functions that either write or read database or package states are not parallelised.
- DML operations on tables with triggers that could fire are not parallelised.
- DML operations on tables or schema objects containing object types, LONGs, or LOB data types are not parallelised.

Parallel DML mode can be modified only between committed transactions. Issuing this command following an uncommitted transaction generates an error.

Parallel DML Example 1

Issue the following statement to enable parallel DML mode for the current session:

```
ALTER SESSION ENABLE PARALLEL DML;
```

Parallel DML Example 2

The following example modifies the current session to check all deferrable constraints immediately following each DML statement:

```
ALTER SESSION SET CONSTRAINTS IMMEDIATE;
```

Parallel DML Example 3

The following statement modifies the current session to allow inserts into local index partitions marked as unusable:

```
ALTER SESSION SET SKIP_UNUSABLE_INDEXES=TRUE;
```

7.2 UPDATE

UPDATE

Update using DML is used for changing data in a table.

For the following tests, the STUDENT table will be used for updating.

The original student table is as follows:

```
SQL> select * from student;

       SID FNAME                LNAME                       AGE HCOLOR         HEIGHT
---------- -------------------- -------------------- ---------- ---------- ----------
         1 bob                  smart                        15 black             170
         2 sam                  taps                         21 blond             160
         3 craig                stone                        34 brown             150
         4 tom                  spat                         66 red               166
         5 pat                  stone                        45 brown             179
         6 tim                  cray                         55 black             166
         7 paul                 craz                         77 grey              187
         8 dawn                 mal                          66 brown             130
         9 jack                 jones                        55 blond             180
        10 mel                  hack                         12 green             190
        11 jones                Ng                           23 black             180
        12 stan                 jack                         33 white             170
        14 rick                 sam                          25 brown             160
        15 paul                 jones                        22 black             188
        16 james                long                         15 brown             155
        17 jack                 brown                        26 none              160
        18 stan                 short                        55 blond             178
        19 wang                 chris                        21 grey              155
        20 john                 black                        44 black             156
       999 Paula                Marsh                        16 Brown             177
       989 pete                 line                         19 Black             187

21 rows selected.
```

There is a hand full of ways in which a table can be updated. The following is some of the techniques used to update tables.

The following updates the record(s) in the age column to NULL when the same record's hcolor is none. This is an example of when a record needs to be updated but there are only unique records in another column.

Update example 1 - serial

```
SQL> update student
  2  set age = NULL
  3  where hcolor = 'none';

1 row updated.
```

After updating the record, the following student was recorded.

```
SQL> select * from student;

       SID FNAME                LNAME                       AGE HCOLOR         HEIGHT
---------- -------------------- -------------------- ---------- ---------- ----------
         1 bob                  smart                        15 black             170
         2 sam                  taps                         21 blond             160
         3 craig                stone                        34 brown             150
         4 tom                  spat                         66 red               166
         5 pat                  stone                        45 brown             179
         6 tim                  cray                         55 black             166
         7 paul                 craz                         77 grey              187
         8 dawn                 mal                          66 brown             130
         9 jack                 jones                        55 blond             180
        10 mel                  hack                         12 green             190
        11 jones                Ng                           23 black             180
        12 stan                 jack                         33 white             170
        14 rick                 sam                          25 brown             160
        15 paul                 jones                        22 black             188
        16 james                long                         15 brown             155
        17 jack                 brown                           none              160
        18 stan                 short                        55 blond             178
        19 wang                 chris                        21 grey              155
        20 john                 black                        44 black             156
       999 Paula                Marsh                        16 Brown             177
       989 pete                 line                         19 Black             187

21 rows selected.
```

The record where once was an age, is now changed to NULL. This is only possible if that column did not have a not null when the table was created.

Update example 1 - parallel

The table were changed back to the original state before performing the parallel version.

```
SQL> update /*+ parallel (student, 2) */ student
  2  set age = NULL
  3  where hcolor = 'none';

1 row updated.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 UPDATE STATEMENT
         1 UPDATE
         2 TABLE ACCESS
```

After updating the record, the following student was recorded.

```
SQL> select * from student;

       SID FNAME                LNAME                       AGE HCOLOR         HEIGHT
---------- -------------------- -------------------- ---------- ---------- ----------
         1 bob                  smart                        15 black             170
         2 sam                  taps                         21 blond             160
         3 craig                stone                        34 brown             150
         4 tom                  spat                         66 red               166
         5 pat                  stone                        45 brown             179
         6 tim                  cray                         55 black             166
         7 paul                 craz                         77 grey              187
         8 dawn                 mal                          66 brown             130
         9 jack                 jones                        55 blond             180
        10 mel                  hack                         12 green             190
        11 jones                Ng                           23 black             180
        12 stan                 jack                         33 white             170
        14 rick                 sam                          25 brown             160
        15 paul                 jones                        22 black             188
        16 james                long                         15 brown             155
        17 jack                 brown                           none              160
        18 stan                 short                        55 blond             178
        19 wang                 chris                        21 grey              155
        20 john                 black                        44 black             156
       999 Paula                Marsh                        16 Brown             177
       989 pete                 line                         19 Black             187

21 rows selected.
```

The record where once was an age, is now changed to NULL. This is only possible if that column did not have a not null when the table was created.

Update example 2 - serial

The following updates the student table and updates more than one column.
The student table original state is after the example prior to this one.

```
SQL> update student
  2  set lname = 'Manns',
  3  hcolor = 'Pink',
  4  height = '197'
  5  where fname = 'bob';

1 row updated.
```

```
SQL> select * from student;
       SID FNAME                LNAME                       AGE HCOLOR         HEIGHT
---------- -------------------- -------------------- ---------- ---------- ----------
         1 bob                  Manns                        15 Pink              197
         2 sam                  taps                         21 blond             160
         3 craig                stone                        34 brown             150
         4 tom                  spat                         66 red               166
         5 pat                  stone                        45 brown             179
         6 tim                  cray                         55 black             166
         7 paul                 craz                         77 grey              187
         8 dawn                 mal                          66 brown             130
         9 jack                 jones                        55 blond             180
        10 mel                  hack                         12 green             190
        11 jones                Ng                           23 black             180
        12 stan                 jack                         33 white             170
        14 rick                 sam                          25 brown             160
        15 paul                 jones                        22 black             188
        16 james                long                         15 brown             155
        17 jack                 brown                           none              160
        18 stan                 short                        55 blond             178
        19 wang                 chris                        21 grey              155
        20 john                 black                        44 black             156
       999 Paula                Marsh                        16 Brown             177
       989 pete                 line                         19 Black             187

21 rows selected.
```

Update example 2 - parallel

The following updates the student table and updates more than one column. The student table original state is after the example prior to this one.

```
SQL> update /*+ parallel (student, 2) */ student
  2  set lname = 'Manns',
  3  hcolor = 'Pink',
  4  height = '197'
  5  where fname = 'bob';

1 row updated.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 UPDATE STATEMENT
         1 UPDATE
         2 TABLE ACCESS
```

```
STATISTIC                      LAST_QUERY SESSION_TOTAL
------------------------------ ---------- -------------
Queries Parallelized                    1             1
DML Parallelized                        0             0
DFO Trees                               1             1
Server Threads                          7             0
Allocation Height                       7             0
Allocation Width                        1             0
Local Msgs Sent                        16            16
Distr Msgs Sent                         0             0
Local Msgs Recv'd                      16            16
Distr Msgs Recv'd                       0             0
```

```
SQL> select * from student;
       SID FNAME                LNAME                       AGE HCOLOR         HEIGHT
---------- -------------------- -------------------- ---------- ---------- ----------
         1 bob                  Manns                        15 Pink              197
         2 sam                  taps                         21 blond             160
         3 craig                stone                        34 brown             150
         4 tom                  spat                         66 red               166
         5 pat                  stone                        45 brown             179
         6 tim                  cray                         55 black             166
         7 paul                 craz                         77 grey              187
         8 dawn                 mal                          66 brown             130
         9 jack                 jones                        55 blond             180
        10 mel                  hack                         12 green             190
        11 jones                Ng                           23 black             180
        12 stan                 jack                         33 white             170
        14 rick                 sam                          25 brown             160
        15 paul                 jones                        22 black             188
        16 james                long                         15 brown             155
        17 jack                 brown                           none              160
        18 stan                 short                        55 blond             178
        19 wang                 chris                        21 grey              155
        20 john                 black                        44 black             156
       999 Paula                Marsh                        16 Brown             177
       989 pete                 line                         19 Black             187

21 rows selected.
```

Update example 3 - serial

Now suppose that the height of each student is required to be in metres rather than centermeters, we would need to convert it. To do this, we perform the following operation:

```
SQL> update student
  2  set height = height / 100;

22 rows updated.
```

There were no WHERE clause here to select certain records, thus all the records in the table, for that column is updated.

```
SQL> select * from student;

       SID FNAME                LNAME                       AGE HCOLOR         HEIGHT
---------- -------------------- -------------------- ---------- ---------- ----------
         1 bob                  smart                        15 black             1.7
         2 sam                  taps                         21 blond             1.6
         3 craig                stone                        34 brown             1.5
         4 tom                  spat                         66 red              1.66
         5 pat                  stone                        45 brown            1.79
         6 tim                  cray                         55 black            1.66
         7 paul                 craz                         77 grey             1.87
         8 dawn                 mal                          66 brown             1.3
         9 jack                 jones                        55 blond             1.8
        10 mel                  hack                         12 green             1.9
        11 jones                Ng                           23 black             1.8
        12 stan                 jack                         33 white             1.7
        21 craig                stap                         44 green           1.794
        14 rick                 sam                          25 brown             1.6
        15 paul                 jones                        22 black            1.88
        16 james                long                         15 brown            1.55
        17 jack                 brown                        26 none              1.6
        18 stan                 short                        55 blond            1.78
        19 wang                 chris                        21 grey             1.55
        20 john                 black                        44 black            1.56
       999 Paula                Marsh                        16 Brown            1.77
       989 pete                 line                         19 Black            1.87

22 rows selected.
```

Update example 3 - parallel

Now suppose that the height of each student is required to be in metres rather than centermeters, we would need to convert it. To do this, we perform the following operation:

```
SQL> update /*+ parallel (student, 2) */ student
  2  set height = height / 100;

22 rows updated.
```

There were no WHERE clause here to select certain records, thus all the records in the table, for that column is updated.

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 UPDATE STATEMENT
         1 UPDATE
         2 TABLE ACCESS
```

```
SQL> select * from v$pq_sesstat;

STATISTIC                      LAST_QUERY SESSION_TOTAL
------------------------------ ---------- -------------
Queries Parallelized                    1             1
DML Parallelized                        0             0
DFO Trees                               1             1
Server Threads                          2             0
Allocation Height                       2             0
Allocation Width                        1             0
Local Msgs Sent                        21            21
Distr Msgs Sent                         0             0
Local Msgs Recv'd                      21            21
Distr Msgs Recv'd                       0             0

10 rows selected.
```

```
SQL> select * from student;

       SID FNAME                LNAME                       AGE HCOLOR         HEIGHT
---------- -------------------- -------------------- ---------- ---------- ----------
         1 bob                  smart                        15 black             1.7
         2 sam                  taps                         21 blond             1.6
         3 craig                stone                        34 brown             1.5
         4 tom                  spat                         66 red              1.66
         5 pat                  stone                        45 brown            1.79
         6 tim                  cray                         55 black            1.66
         7 paul                 craz                         77 grey             1.87
         8 dawn                 mal                          66 brown             1.3
         9 jack                 jones                        55 blond             1.8
        10 mel                  hack                         12 green             1.9
        11 jones                Ng                           23 black             1.8
        12 stan                 jack                         33 white             1.7
        21 craig                stap                         44 green           1.794
        14 rick                 sam                          25 brown             1.6
        15 paul                 jones                        22 black            1.88
        16 james                long                         15 brown            1.55
        17 jack                 brown                        26 none              1.6
        18 stan                 short                        55 blond            1.78
        19 wang                 chris                        21 grey             1.55
        20 john                 black                        44 black            1.56
       999 Paula                Marsh                        16 Brown            1.77
       989 pete                 line                         19 Black            1.87

22 rows selected.
```

Update example 4 - serial

The following updates a record in the student table accessible through the database link bus4410.

```
SQL> update s12288624.student@bus4410
  2  set sid = 600
  3  where fname = 'pete';

1 row updated.
```

```
SQL> select * from student;

       SID FNAME                LNAME                       AGE HCOLOR         HEIGHT
---------- -------------------- -------------------- ---------- ---------- ----------
         1 bob                  smart                        15 black             1.7
         2 sam                  taps                         21 blond             1.6
         3 craig                stone                        34 brown             1.5
         4 tom                  spat                         66 red              1.66
         5 pat                  stone                        45 brown            1.79
         6 tim                  cray                         55 black            1.66
         7 paul                 craz                         77 grey             1.87
         8 dawn                 mal                          66 brown             1.3
         9 jack                 jones                        55 blond             1.8
        10 mel                  hack                         12 green             1.9
        11 jones                Ng                           23 black             1.8
        12 stan                 jack                         33 white             1.7
        21 craig                stap                         44 green           1.794
        14 rick                 sam                          25 brown             1.6
        15 paul                 jones                        22 black            1.88
        16 james                long                         15 brown            1.55
        17 jack                 brown                        26 none              1.6
        18 stan                 short                        55 blond            1.78
        19 wang                 chris                        21 grey             1.55
        20 john                 black                        44 black            1.56
       999 Paula                Marsh                        16 Brown            1.77
       600 pete                 line                         19 Black            1.87

22 rows selected.
```

Update example 4 - parallel

The following updates a record in the student table accessible through the database link bus4410.

```
SQL> update /*+ parallel s12288624.student@bus4410 */
  2  s12288624.student@bus4410
  3  set sid = 600
  4  where fname = 'pete';

1 row updated.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 UPDATE STATEMENT
         1 UPDATE
         2 TABLE ACCESS
```

```
STATISTIC                      LAST_QUERY SESSION_TOTAL
------------------------------ ---------- -------------
Queries Parallelized                    1             1
DML Parallelized                        0             0
DFO Trees                               1             1
Server Threads                          2             0
Allocation Height                       2             0
Allocation Width                        1             0
Local Msgs Sent                         1             1
Distr Msgs Sent                         0             0
Local Msgs Recv'd                       1             1
Distr Msgs Recv'd                       0             0
```

```
SQL> select * from student;

       SID FNAME                LNAME                       AGE HCOLOR         HEIGHT
---------- -------------------- -------------------- ---------- ---------- ----------
         1 bob                  smart                        15 black             1.7
         2 sam                  taps                         21 blond             1.6
         3 craig                stone                        34 brown             1.5
         4 tom                  spat                         66 red              1.66
         5 pat                  stone                        45 brown            1.79
         6 tim                  cray                         55 black            1.66
         7 paul                 craz                         77 grey             1.87
         8 dawn                 mal                          66 brown             1.3
         9 jack                 jones                        55 blond             1.8
        10 mel                  hack                         12 green             1.9
        11 jones                Ng                           23 black             1.8
        12 stan                 jack                         33 white             1.7
        21 craig                stap                         44 green           1.794
        14 rick                 sam                          25 brown             1.6
        15 paul                 jones                        22 black            1.88
        16 james                long                         15 brown            1.55
        17 jack                 brown                        26 none              1.6
        18 stan                 short                        55 blond            1.78
        19 wang                 chris                        21 grey             1.55
        20 john                 black                        44 black            1.56
       999 Paula                Marsh                        16 Brown            1.77
       600 pete                 line                         19 Black            1.87

22 rows selected.
```

Update example 5 - serial

This example shows the following syntactic constructs of the UPDATE command:

- both forms of the SET clause together in a single statement
- a correlated sub query, and
- a WHERE clause to limit the updated rows

The below update statement updates only those student(s) who has the last name 'Line', sets the age to 1.5 x the average ages and 2 x the average heights from student table. And updates the records only if first name were also found in the customer table.

```
SQL> update student
       set sid =
              (select sid
               from student
               where lname = 'line'),
              (age, height) =
                     (select 1.5*avg(age), 2*avg(height)
                      from student)
       where fname in
              (select first
               from customer
               where first = 'pete');

1 row updated.
```

```
SQL> select * from student;

       SID FNAME                LNAME                       AGE HCOLOR         HEIGHT
---------- -------------------- -------------------- ---------- ---------- ----------
         1 bob                  smart                        15 black             1.7
         2 sam                  taps                         21 blond             1.6
         3 craig                stone                        34 brown             1.5
         4 tom                  spat                         66 red              1.66
         5 pat                  stone                        45 brown            1.79
         6 tim                  cray                         55 black            1.66
         7 paul                 craz                         77 grey             1.87
         8 dawn                 mal                          66 brown             1.3
         9 jack                 jones                        55 blond             1.8
        10 mel                  hack                         12 green             1.9
        11 jones                Ng                           23 black             1.8
        12 stan                 jack                         33 white             1.7
        21 craig                stap                         44 green           1.794
        14 rick                 sam                          25 brown             1.6
        15 paul                 jones                        22 black            1.88
        16 james                long                         15 brown            1.55
        17 jack                 brown                        26 none              1.6
        18 stan                 short                        55 blond            1.78
        19 wang                 chris                        21 grey             1.55
        20 john                 black                        44 black            1.56
       999 Paula                Marsh                        16 Brown            1.77
       600 pete                 line                 53.7954545 Black      3.38490909

22 rows selected.
```

Update example 5 - parallel

This example shows the following syntactic constructs of the UPDATE command:

- both forms of the SET clause together in a single statement
- a correlated sub query, and
- a WHERE clause to limit the updated rows

The below update statement updates only those student(s) who has the last name ‘Line’, sets the age to 1.5 x the average ages and 2 x the average heights from student table. And updates the records only if first name were also found in the customer table.

```
SQL> update /*+ parallel (student, 2) */ student
  2   set sid =
  3         (select sid
  4          from student
  5          where lname = 'line'),
  6         (age, height) =
  7                (select 1.5*avg(age), 2*avg(height)
  8                 from student)
  9   where fname in
 10         (select first
 11          from customer
 12          where first = 'pete');

1 row updated.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 UPDATE STATEMENT
         1 UPDATE
         2 TABLE ACCESS
```

```
STATISTIC                      LAST_QUERY SESSION_TOTAL
------------------------------ ---------- -------------
Queries Parallelized                    1             1
DML Parallelized                        0             0
DFO Trees                               1             1
Server Threads                          2             0
Allocation Height                       2             0
Allocation Width                        1             0
Local Msgs Sent                         1             1
Distr Msgs Sent                         0             0
Local Msgs Recv'd                       1             1
Distr Msgs Recv'd                       0             0
```

```
SQL> select * from student;

       SID FNAME                LNAME                       AGE HCOLOR         HEIGHT
---------- -------------------- -------------------- ---------- ---------- ----------
         1 bob                  smart                        15 black             1.7
         2 sam                  taps                         21 blond             1.6
         3 craig                stone                        34 brown             1.5
         4 tom                  spat                         66 red              1.66
         5 pat                  stone                        45 brown            1.79
         6 tim                  cray                         55 black            1.66
         7 paul                 craz                         77 grey             1.87
         8 dawn                 mal                          66 brown             1.3
         9 jack                 jones                        55 blond             1.8
        10 mel                  hack                         12 green             1.9
        11 jones                Ng                           23 black             1.8
        12 stan                 jack                         33 white             1.7
        21 craig                stap                         44 green           1.794
        14 rick                 sam                          25 brown             1.6
        15 paul                 jones                        22 black            1.88
        16 james                long                         15 brown            1.55
        17 jack                 brown                        26 none              1.6
        18 stan                 short                        55 blond            1.78
        19 wang                 chris                        21 grey             1.55
        20 john                 black                        44 black            1.56
       999 Paula                Marsh                        16 Brown            1.77
       600 pete                 line                 53.7954545 Black      3.38490909

22 rows selected.
```

7.3 INSERT

INSERT

Insert command is to insert data records into the tables.

In this example we use the COURSE table.
The original table to begin with:

```
SQL> select * from course;

       CID CALLID     CNAME                       CCREDIT
---------- ---------- ------------------------- ----------
         1 MIS 101    Intro. to Info. Systems            3
         2 MIS 301    Systems Analysis                   3
         3 MIS 441    Database Management                3
         4 CS 155     Programming in C++                 3
         5 MIS 451    Client/Server Systems              3
```

Insert example 1 – serial

The normal and simplest method of inserting a record into a table is doing the following:

```
SQL> insert into course values
  2  (6, 'BUS 5000', 'Reading Unit', 6);

1 row created.
```

```
SQL> select * from course;

       CID CALLID     CNAME                        CCREDIT
---------- ---------- ------------------------- ----------
         1 MIS 101    Intro. to Info. Systems            3
         2 MIS 301    Systems Analysis                   3
         3 MIS 441    Database Management                3
         4 CS 155     Programming in C++                 3
         5 MIS 451    Client/Server Systems              3
         6 BUS 5000   Reading Unit                       6

6 rows selected.
```

Insert example 1 – parallel

The normal and simplest method of inserting a record into a table is doing the following:

```
SQL> insert /*+ parallel (course, 2) */ into course values
  2  (6, 'BUS 5000', 'Reading Unit', 6);

1 row created.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 UPDATE STATEMENT
         1 UPDATE
         2 TABLE ACCESS
```

```
STATISTIC                      LAST_QUERY SESSION_TOTAL
------------------------------ ---------- -------------
Queries Parallelized                    0             0
DML Parallelized                        0             0
DFO Trees                               0             0
Server Threads                          0             0
Allocation Height                       0             0
Allocation Width                        0             0
Local Msgs Sent                         0             0
Distr Msgs Sent                         0             0
Local Msgs Recv'd                       0             0
Distr Msgs Recv'd                       0             0

10 rows selected.
```

```
SQL> select * from course;

       CID CALLID     CNAME                        CCREDIT
---------- ---------- ------------------------- ----------
         1 MIS 101    Intro. to Info. Systems            3
         2 MIS 301    Systems Analysis                   3
         3 MIS 441    Database Management                3
         4 CS 155     Programming in C++                 3
         5 MIS 451    Client/Server Systems              3
         6 BUS 5000   Reading Unit                       6

6 rows selected.
```

Insert example 2 – sequence - serial

In this example, we create a sequence, which allows ORACLE to increment the CID to the next value when a new record is inserted.

To create a sequence:

```
SQL> create sequence cid_sequence
  2  start with 7;

Sequence created.
```

```
SQL> insert into course(cid, callid, cname, ccredit)
  2  values (cid_sequence.nextval, 'BUS4020', 'Trading systems', 6);

1 row created.
```

```
SQL> select * from course;

       CID CALLID     CNAME                        CCREDIT
---------- ---------- ------------------------- ----------
         1 MIS 101    Intro. to Info. Systems            3
         2 MIS 301    Systems Analysis                   3
         3 MIS 441    Database Management                3
         4 CS 155     Programming in C++                 3
         5 MIS 451    Client/Server Systems              3
         6 BUS 5000   Reading Unit                       6
         7 BUS4020    Trading systems                    6

7 rows selected.
```

The above CID of 7 is automatically inserted as the next value.

Insert example 2 – sequence - parallel

In this example, we create a sequence, which allows ORACLE to increment the CID to the next value when a new record is inserted.

To create a sequence:

```
SQL> create sequence cid_sequence
  2  start with 7;

Sequence created.
```

```
SQL> insert /*+ parallel (course, 2) */
  2  into course(cid, callid, cname, ccredit)
  3  values (cid_sequence.nextval, 'BUS4020', 'Trading systems', 6);

1 row created.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 UPDATE STATEMENT
         1 UPDATE
         2 TABLE ACCESS
```

```
SQL> select * from v$pq_sesstat;

STATISTIC                      LAST_QUERY SESSION_TOTAL
------------------------------ ---------- -------------
Queries Parallelized                    0             1
DML Parallelized                        0             0
DFO Trees                               0             0
Server Threads                          0             0
Allocation Height                       0             0
Allocation Width                        1             0
Local Msgs Sent                         0             0
Distr Msgs Sent                         0             0
Local Msgs Recv'd                       0             0
Distr Msgs Recv'd                       0             0

10 rows selected.
```

```
SQL> select * from course;

       CID CALLID     CNAME                        CCREDIT
---------- ---------- ------------------------- ----------
         1 MIS 101    Intro. to Info. Systems            3
         2 MIS 301    Systems Analysis                   3
         3 MIS 441    Database Management                 3
         4 CS 155     Programming in C++                 3
         5 MIS 451    Client/Server Systems              3
         6 BUS 5000   Reading Unit                       6
         7 BUS4020    Trading systems                    6

7 rows selected.
```

The above CID of 7 is automatically inserted as the next value.

Insert example 3 - serial

The following uses the column names to bound the inserts.

```
SQL> insert into course (cid, callid, cname, ccredit)
  2  values(8, 'DGS1111', 'Digital systems 1', 6);

1 row created.
```

```
SQL> select * from course;

       CID CALLID     CNAME                        CCREDIT
---------- ---------- ------------------------- ----------
         1 MIS 101    Intro. to Info. Systems            3
         2 MIS 301    Systems Analysis                   3
         3 MIS 441    Database Management                 3
         4 CS 155     Programming in C++                 3
         5 MIS 451    Client/Server Systems              3
         6 BUS 5000   Reading Unit                       6
         7 BUS4020    Trading systems                    6
         8 DGS1111    Digital systems 1                  6

8 rows selected.
```

Insert example 3 - parallel

The following uses the column names to bound the inserts.

```
SQL> insert into course (cid, callid, cname, ccredit)
  2  values(8, 'DGS1111', 'Digital systems 1', 6);

1 row created.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 UPDATE STATEMENT
         1 UPDATE
         2 TABLE ACCESS
```

```
SQL> select * from v$pq_sesstat;

STATISTIC                      LAST_QUERY SESSION_TOTAL
------------------------------ ---------- -------------
Queries Parallelized                    0             1
DML Parallelized                        0             0
DFO Trees                               0             0
Server Threads                          0             0
Allocation Height                       0             0
Allocation Width                        0             0
Local Msgs Sent                         1             0
Distr Msgs Sent                         0             0
Local Msgs Recv'd                       0             0
Distr Msgs Recv'd                       0             0

10 rows selected.
```

```
SQL> select * from course;

       CID CALLID     CNAME                       CCREDIT
---------- ---------- ------------------------- ----------
         1 MIS 101    Intro. to Info. Systems            3
         2 MIS 301    Systems Analysis                   3
         3 MIS 441    Database Management                3
         4 CS 155     Programming in C++                 3
         5 MIS 451    Client/Server Systems              3
         6 BUS 5000   Reading Unit                       6
         7 BUS4020    Trading systems                    6
         8 DGS1111    Digital systems 1                  6

8 rows selected.
```

Insert example 4 – NULL inserts - serial

A NULL value is inserted to CNAME (course name).

```
SQL> insert into course (cid, callid, cname, ccredit)
  2  values (9, 'BUS8000', NULL, 6);

1 row created.
```

```
SQL> select * from course;

       CID CALLID     CNAME                        CCREDIT
---------- ---------- ------------------------- ----------
         1 MIS 101    Intro. to Info. Systems            3
         2 MIS 301    Systems Analysis                   3
         3 MIS 441    Database Management                3
         4 CS 155     Programming in C++                 3
         5 MIS 451    Client/Server Systems              3
         6 BUS 5000   Reading Unit                       6
         7 BUS4020    Trading systems                    6
         8 DGS1111    Digital systems 1                  6
         9 BUS8000                                       6

9 rows selected.
```

Insert example 4 – NULL inserts - parallel

A NULL value is inserted to CNAME (course name).

```
SQL> insert into course (cid, callid, cname, ccredit)
  2  values (9, 'BUS8000', NULL, 6);

1 row created.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 UPDATE STATEMENT
         1 UPDATE
         2 TABLE ACCESS
```

```
SQL> select * from v$pq_sesstat;

STATISTIC                      LAST_QUERY SESSION_TOTAL
------------------------------ ---------- -------------
Queries Parallelized                    0             1
DML Parallelized                        0             0
DFO Trees                               0             0
Server Threads                          2             2
Allocation Height                       2             2
Allocation Width                        0             0
Local Msgs Sent                         1             1
Distr Msgs Sent                         1             1
Local Msgs Recv'd                       0             0
Distr Msgs Recv'd                       0             0

10 rows selected.
```

```
SQL> select * from course;

       CID CALLID     CNAME                        CCREDIT
---------- ---------- ------------------------- ----------
         1 MIS 101    Intro. to Info. Systems            3
         2 MIS 301    Systems Analysis                   3
         3 MIS 441    Database Management                3
         4 CS 155     Programming in C++                 3
         5 MIS 451    Client/Server Systems              3
         6 BUS 5000   Reading Unit                       6
         7 BUS4020    Trading systems                    6
         8 DGS1111    Digital systems 1                  6
         9 BUS8000                                       6

9 rows selected.
```

Insert example 5 – Select into - serial

A table of top students is created to store the top students that achieved high scores.

```
SQL> create table top_students(
  2  sid number,
  3  sfname varchar2(15),
  4  slname varchar2(15),
  5  csecid number,
  6  grade varchar2(2));

Table created.
```

The table is filled with the selected students with grades A or B.

```
SQL> insert into top_students
  2  select s.sid, s.sfname, s.slname, e.csecid, e.grade
  3  from student s, enrollment e
  4  where e.sid = s.sid AND
  5  e.grade = 'A' OR
  6  e.grade = 'B';

33 rows created.
```

The students together with the course section ID is recorded

```
SQL> select * from top_students;

       SID SFNAME          SLNAME              CSECID GR
---------- --------------- --------------- ---------- --
       100 Sarah           Miller                1000 A
       100 Sarah           Miller                1003 A
       100 Sarah           Miller                1005 B
       101 Brian           Umato                 1005 B
       102 Daniel          Black                 1005 B
       103 Amanda          Mobley                1005 B
       104 Ruben           Sanchez               1005 B
       105 Michael         Connoly               1005 B
       100 Sarah           Miller                1008 B
       101 Brian           Umato                 1008 B
       102 Daniel          Black                 1008 B
       103 Amanda          Mobley                1008 B
       104 Ruben           Sanchez               1008 B
       105 Michael         Connoly               1008 B
       100 Sarah           Miller                1004 B
       101 Brian           Umato                 1004 B
       102 Daniel          Black                 1004 B
       103 Amanda          Mobley                1004 B
       104 Ruben           Sanchez               1004 B
       105 Michael         Connoly               1004 B
       101 Brian           Umato                 1005 A
       100 Sarah           Miller                1008 B
       101 Brian           Umato                 1008 B
       102 Daniel          Black                 1008 B
       103 Amanda          Mobley                1008 B
       104 Ruben           Sanchez               1008 B
       105 Michael         Connoly               1008 B
       100 Sarah           Miller                1000 B
       101 Brian           Umato                 1000 B
       102 Daniel          Black                 1000 B
       103 Amanda          Mobley                1000 B
       104 Ruben           Sanchez               1000 B
       105 Michael         Connoly               1000 B

33 rows selected.
```

Insert example 5 – Select into - parallel

A table of top students is created to store the top students that achieved high scores.

```
SQL> create table top_students(
  2  sid number,
  3  sfname varchar2(15),
  4  slname varchar2(15),
  5  csecid number,
  6  grade varchar2(2));

Table created.
```

The table is filled with the selected students with grades A or B.

```
SQL> insert into top_students
  2  select s.sid, s.sfname, s.slname, e.csecid, e.grade
  3  from student s, enrollment e
  4  where e.sid = s.sid AND
  5  e.grade = 'A' OR
  6  e.grade = 'B';

33 rows created.
```

```
SQL> select * from v$pq_sesstat;

STATISTIC                      LAST_QUERY SESSION_TOTAL
------------------------------ ---------- -------------
Queries Parallelized                    1             3
DML Parallelized                        0             0
DFO Trees                               1             3
Server Threads                          2             0
Allocation Height                       2             0
Allocation Width                        1             0
Local Msgs Sent                        10            42
Distr Msgs Sent                         0             0
Local Msgs Recv'd                      10            42
Distr Msgs Recv'd                       0             0

10 rows selected.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 UPDATE STATEMENT
         1 UPDATE
         2 TABLE ACCESS
```

The students together with the course section ID is recorded

```
SQL> select * from top_students;

       SID SFNAME          SLNAME              CSECID GR
---------- --------------- --------------- ---------- --
       100 Sarah           Miller                1000 A
       100 Sarah           Miller                1003 A
       100 Sarah           Miller                1005 B
       101 Brian           Umato                 1005 B
       102 Daniel          Black                 1005 B
       103 Amanda          Mobley                1005 B
       104 Ruben           Sanchez               1005 B
       105 Michael         Connoly               1005 B
       100 Sarah           Miller                1008 B
       101 Brian           Umato                 1008 B
       102 Daniel          Black                 1008 B
       103 Amanda          Mobley                1008 B
       104 Ruben           Sanchez               1008 B
       105 Michael         Connoly               1008 B
       100 Sarah           Miller                1004 B
       101 Brian           Umato                 1004 B
       102 Daniel          Black                 1004 B
       103 Amanda          Mobley                1004 B
       104 Ruben           Sanchez               1004 B
       105 Michael         Connoly               1004 B
       101 Brian           Umato                 1005 A
       100 Sarah           Miller                1008 B
       101 Brian           Umato                 1008 B
       102 Daniel          Black                 1008 B
       103 Amanda          Mobley                1008 B
       104 Ruben           Sanchez               1008 B
       105 Michael         Connoly               1008 B
       100 Sarah           Miller                1000 B
       101 Brian           Umato                 1000 B
       102 Daniel          Black                 1000 B
       103 Amanda          Mobley                1000 B
       104 Ruben           Sanchez               1000 B
       105 Michael         Connoly               1000 B
```

7.4 DELETE

Delete allows you to remove rows from a table, a partitioned table, a view's base table.

Delete example 1 – all records - serial

The original top_students table content.

```
SQL> select * from top_students;

       SID SFNAME         SLNAME               CSECID GR
---------- -------------- -------------- ---------- --
       100 Sarah          Miller               1000 A
       100 Sarah          Miller               1003 A
       100 Sarah          Miller               1005 B
       101 Brian          Umato                1005 B
       102 Daniel         Black                1005 B
       103 Amanda         Mobley               1005 B
       104 Ruben          Sanchez              1005 B
       105 Michael        Connoly              1005 B
       100 Sarah          Miller               1008 B
       101 Brian          Umato                1008 B
       102 Daniel         Black                1008 B
       103 Amanda         Mobley               1008 B
       104 Ruben          Sanchez              1008 B
       105 Michael        Connoly              1008 B
       100 Sarah          Miller               1004 B
       101 Brian          Umato                1004 B
       102 Daniel         Black                1004 B
       103 Amanda         Mobley               1004 B
       104 Ruben          Sanchez              1004 B
       105 Michael        Connoly              1004 B
       101 Brian          Umato                1005 A
       100 Sarah          Miller               1008 B
       101 Brian          Umato                1008 B
       102 Daniel         Black                1008 B
       103 Amanda         Mobley               1008 B
       104 Ruben          Sanchez              1008 B
       105 Michael        Connoly              1008 B
       100 Sarah          Miller               1000 B
       101 Brian          Umato                1000 B
       102 Daniel         Black                1000 B
       103 Amanda         Mobley               1000 B
       104 Ruben          Sanchez              1000 B
       105 Michael        Connoly              1000 B

33 rows selected.
```

The following deletes all records in the top_student table.

```
SQL> delete from top_students;

33 rows deleted.
```

No more records are stored in the table anymore.

```
SQL>  select * from top_students;

no rows selected
```

Delete example 1 – all records - parallel

The original top_students table content.

```
SQL> select * from top_students;

       SID SFNAME          SLNAME              CSECID GR
---------- --------------- --------------- ---------- --
       100 Sarah           Miller                1000 A
       100 Sarah           Miller                1003 A
       100 Sarah           Miller                1005 B
       101 Brian           Umato                 1005 B
       102 Daniel          Black                 1005 B
       103 Amanda          Mobley                1005 B
       104 Ruben           Sanchez               1005 B
       105 Michael         Connoly               1005 B
       100 Sarah           Miller                1008 B
       101 Brian           Umato                 1008 B
       102 Daniel          Black                 1008 B
       103 Amanda          Mobley                1008 B
       104 Ruben           Sanchez               1008 B
       105 Michael         Connoly               1008 B
       100 Sarah           Miller                1004 B
       101 Brian           Umato                 1004 B
       102 Daniel          Black                 1004 B
       103 Amanda          Mobley                1004 B
       104 Ruben           Sanchez               1004 B
       105 Michael         Connoly               1004 B
       101 Brian           Umato                 1005 A
       100 Sarah           Miller                1008 B
       101 Brian           Umato                 1008 B
       102 Daniel          Black                 1008 B
       103 Amanda          Mobley                1008 B
       104 Ruben           Sanchez               1008 B
       105 Michael         Connoly               1008 B
       100 Sarah           Miller                1000 B
       101 Brian           Umato                 1000 B
       102 Daniel          Black                 1000 B
       103 Amanda          Mobley                1000 B
       104 Ruben           Sanchez               1000 B
       105 Michael         Connoly               1000 B

33 rows selected.
```

The following deletes all records in the top_student table.

```
SQL> delete /*+ parallel (top_students, 2) */ from top_students;

33 rows deleted.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 UPDATE STATEMENT
         1 UPDATE
         2 TABLE ACCESS
         0 INSERT STATEMENT
         0 DELETE STATEMENT
         1 DELETE
         2 TABLE ACCESS

7 rows selected.
```

```
SQL> select * from v$pq_sesstat;

STATISTIC                      LAST_QUERY SESSION_TOTAL
------------------------------ ---------- -------------
Queries Parallelized                    0             4
DML Parallelized                        0             0
DFO Trees                               0             4
Server Threads                          0             0
Allocation Height                       0             0
Allocation Width                        0             0
Local Msgs Sent                         0            52
Distr Msgs Sent                         0             0
Local Msgs Recv'd                       0            52
Distr Msgs Recv'd                       0             0

10 rows selected.
```

No more records are stored in the table anymore.

```
SQL>  select * from top_students;

no rows selected
```

Delete example 2 - serial

The top_student table is filled with data once more.

```
SQL> insert into top_students
  2  select s.sid, s.sfname, s.slname, e.csecid, e.grade
  3  from student s, enrollment e
  4  where e.sid = s.sid AND
  5  e.grade = 'A' OR
  6  e.grade = 'B';

33 rows created.
```

```
SQL> select * from top_students;

       SID SFNAME          SLNAME              CSECID GR
---------- --------------- --------------- ---------- --
       100 Sarah           Miller                1000 A
       100 Sarah           Miller                1003 A
       100 Sarah           Miller                1005 B
       101 Brian           Umato                 1005 B
       102 Daniel          Black                 1005 B
       103 Amanda          Mobley                1005 B
       104 Ruben           Sanchez               1005 B
       105 Michael         Connoly               1005 B
       100 Sarah           Miller                1008 B
       101 Brian           Umato                 1008 B
       102 Daniel          Black                 1008 B
       103 Amanda          Mobley                1008 B
       104 Ruben           Sanchez               1008 B
       105 Michael         Connoly               1008 B
       100 Sarah           Miller                1004 B
       101 Brian           Umato                 1004 B
       102 Daniel          Black                 1004 B
       103 Amanda          Mobley                1004 B
       104 Ruben           Sanchez               1004 B
       105 Michael         Connoly               1004 B
       101 Brian           Umato                 1005 A
       100 Sarah           Miller                1008 B
       101 Brian           Umato                 1008 B
       102 Daniel          Black                 1008 B
       103 Amanda          Mobley                1008 B
       104 Ruben           Sanchez               1008 B
       105 Michael         Connoly               1008 B
       100 Sarah           Miller                1000 B
       101 Brian           Umato                 1000 B
       102 Daniel          Black                 1000 B
       103 Amanda          Mobley                1000 B
       104 Ruben           Sanchez               1000 B
       105 Michael         Connoly               1000 B

33 rows selected.
```

```
SQL> delete from top_students
  2  where sfname = 'Michael' OR
  3  slname = 'Sanchez';

10 rows deleted.
```

Delete example 2 - parallel

The top_student table is filled with data once more.

```
SQL> insert into top_students
  2  select s.sid, s.sfname, s.slname, e.csecid, e.grade
  3  from student s, enrollment e
  4  where e.sid = s.sid AND
  5  e.grade = 'A' OR
  6  e.grade = 'B';

33 rows created.
```

```
SQL> select * from top_students;

       SID SFNAME          SLNAME              CSECID GR
---------- --------------- --------------- ---------- --
       100 Sarah           Miller                1000 A
       100 Sarah           Miller                1003 A
       100 Sarah           Miller                1005 B
       101 Brian           Umato                 1005 B
       102 Daniel          Black                 1005 B
       103 Amanda          Mobley                1005 B
       104 Ruben           Sanchez               1005 B
       105 Michael         Connoly               1005 B
       100 Sarah           Miller                1008 B
       101 Brian           Umato                 1008 B
       102 Daniel          Black                 1008 B
       103 Amanda          Mobley                1008 B
       104 Ruben           Sanchez               1008 B
       105 Michael         Connoly               1008 B
       100 Sarah           Miller                1004 B
       101 Brian           Umato                 1004 B
       102 Daniel          Black                 1004 B
       103 Amanda          Mobley                1004 B
       104 Ruben           Sanchez               1004 B
       105 Michael         Connoly               1004 B
       101 Brian           Umato                 1005 A
       100 Sarah           Miller                1008 B
       101 Brian           Umato                 1008 B
       102 Daniel          Black                 1008 B
       103 Amanda          Mobley                1008 B
       104 Ruben           Sanchez               1008 B
       105 Michael         Connoly               1008 B
       100 Sarah           Miller                1000 B
       101 Brian           Umato                 1000 B
       102 Daniel          Black                 1000 B
       103 Amanda          Mobley                1000 B
       104 Ruben           Sanchez               1000 B
       105 Michael         Connoly               1000 B

33 rows selected.
```

```
SQL> delete /*+ parallel (top_students) */ from top_students
  2  where sfname = 'Michael' OR
  3  slname = 'Sanchez';

10 rows deleted.
```

```
SQL> select * from v$pq_sesstat;

STATISTIC                      LAST_QUERY SESSION_TOTAL
------------------------------ ---------- -------------
Queries Parallelized                    1             5
DML Parallelized                        0             0
DFO Trees                               1             5
Server Threads                          2             0
Allocation Height                       2             0
Allocation Width                        1             0
Local Msgs Sent                        10            62
Distr Msgs Sent                         0             0
Local Msgs Recv'd                      10            62
Distr Msgs Recv'd                       0             0

10 rows selected.
```

The result of this delete:

```
SQL> select * from top_students;

       SID SFNAME          SLNAME              CSECID GR
---------- --------------- --------------- ---------- --
       100 Sarah           Miller                1000 A
       100 Sarah           Miller                1003 A
       100 Sarah           Miller                1005 B
       101 Brian           Umato                 1005 B
       102 Daniel          Black                 1005 B
       103 Amanda          Mobley                1005 B
       100 Sarah           Miller                1008 B
       101 Brian           Umato                 1008 B
       102 Daniel          Black                 1008 B
       103 Amanda          Mobley                1008 B
       100 Sarah           Miller                1004 B
       101 Brian           Umato                 1004 B
       102 Daniel          Black                 1004 B
       103 Amanda          Mobley                1004 B
       101 Brian           Umato                 1005 A
       100 Sarah           Miller                1008 B
       101 Brian           Umato                 1008 B
       102 Daniel          Black                 1008 B
       103 Amanda          Mobley                1008 B
       100 Sarah           Miller                1000 B
       101 Brian           Umato                 1000 B
       102 Daniel          Black                 1000 B
       103 Amanda          Mobley                1000 B

23 rows selected.
```

Delete example 3 - serial

The following deletes all the SID (student id) that is lower than 103 and with grades other than A.

```
SQL> delete from top_students
  2  where sid < 103 AND
  3  grade <> 'A';

15 rows deleted.
```

The result of the delete:

```
SQL> select * from top_students;

       SID SFNAME          SLNAME              CSECID GR
---------- --------------- --------------- ---------- --
       100 Sarah           Miller                1000 A
       100 Sarah           Miller                1003 A
       103 Amanda          Mobley                1005 B
       103 Amanda          Mobley                1008 B
       103 Amanda          Mobley                1004 B
       101 Brian           Umato                 1005 A
       103 Amanda          Mobley                1008 B
       103 Amanda          Mobley                1000 B

8 rows selected.
```

Delete example 3 - parallel

The following deletes all the SID (student id) that is lower than 103 and with grades other than A.

```
SQL> delete /*+ parallel (top_student, 2) */ from top_students
  2  where sid < 103 AND
  3  grade <> 'A';

15 rows deleted.
```

```
SQL> select * from v$pq_sesstat;

STATISTIC                      LAST_QUERY SESSION_TOTAL
------------------------------ ---------- -------------
Queries Parallelized                    0             5
DML Parallelized                        0             0
DFO Trees                               0             5
Server Threads                          0             0
Allocation Height                       0             0
Allocation Width                        0             0
Local Msgs Sent                         0            62
Distr Msgs Sent                         0             0
Local Msgs Recv'd                       0            62
Distr Msgs Recv'd                       0             0

10 rows selected.
```

```
SQL> select id, operation
  2  from plan_table;

        ID OPERATION
---------- ------------------------------
         0 UPDATE STATEMENT
         1 UPDATE
         2 TABLE ACCESS

3 rows selected.
```

The result of the delete:

```
SQL> select * from top_students;

       SID SFNAME          SLNAME              CSECID GR
---------- --------------- --------------- ---------- --
       100 Sarah           Miller                1000 A
       100 Sarah           Miller                1003 A
       103 Amanda          Mobley                1005 B
       103 Amanda          Mobley                1008 B
       103 Amanda          Mobley                1004 B
       101 Brian           Umato                 1005 A
       103 Amanda          Mobley                1008 B
       103 Amanda          Mobley                1000 B
```

8 rows selected.

CHAPTER 8 – EXTENDED UTILITIES

8.1 PARALLEL DATA LOADING

Parallel Data Loading

Oracle's SQL*Loader utility loads data into Oracle tables from external files. The loading of data can be parallel performed with some restrictions. SQL*Loader's parallel support can dramatically reduce the elapsed time needed to perform that load if there is a large amount of data to load into the tables.

SQL*Loader can:

- Load data from multiple input data files of different file types
- Handle fixed-format, delimited-format, and variable-length records
- Manipulate data fields with SQL functions before inserting the data into database columns
- Support a wide range of data types, including DATE, BINARY, PACKED DECIMAL, and ZONED DECIMAL
- Load multiple tables during the same run, loading selected rows into each table
- Handle a single physical record as multiple logical records
- Generate unique, sequential key values in specified columns
- Thoroughly report errors so you can easily adjust and load all records

8.2 SQL*LOADER

SQL*Loader Control File

The control file, written in SQL*Loader data definition language (DDL), specifies how to interpret the data, what tables and columns to insert the data into, and may also include input data file management information.

The data for SQL*Loader to load into an Oracle database must be in files accessible to SQL*Loader. SQL*Loader requires information about the data to be loaded which provides instructions for mapping the input data to columns of a table. These instructions are written in SQL*Loader DDL.

The following are some of the items that are specified in the SQL*Loader control file:

- Specifications for loading logical records into tables
- Field condition specifications
- Column and field specifications
- Data-field position specifications
- Data type specifications
- Bind array size specifications
- Specifications for setting columns to null or zero
- Specifications for loading all-blank fields
- Specifications for trimming blanks and tabs
- Specifications to preserve white space
- Specifications for applying SQL operators to fields

A single DDL statement comprises one or more keywords and the arguments and options that modify that keyword' s functionality. The following example from a control file contains several statements specifying how SQL*Loader is to load the data from an input data file into a table in an Oracle database:

```
LOAD DATA
INFILE 'example.dat'
INTO TABLE emp
(empno         POSITION(01:04)  INTEGER EXTERNAL,
  ename          POSITION(06:15)  CHAR,
  job            POSITION(17:25)  CHAR,
  mgr            POSITION(27:30)  INTEGER EXTERNAL,
  sal            POSITION(32:39)  DECIMAL EXTERNAL,
  comm           POSITION(41:48)  DECIMAL EXTERNAL,
...
```

This example shows the keywords LOAD DATA, INFILE, INTO TABLE, and POSITION.

Due to the limitations of utilities available to students, this SQL*Loader was unavailable for this experiment.

8.3 PARALLEL RECOVERY

Parallel recovery

Parallel recovery can speed up both instance recovery and media recovery. Multiple parallel dslave processes are used to perform recovery operations in parallel recovery. The SMON background process reds the redo logfiles, and the parallel slave processes apply the changes to the data files. Parallel recovery is most useful when several data files on different disks are being recovered.

Recovery requires that the changes be applied to the data files in exactly the same order in which they occurred.
The RECOVERY_PARALLELISM initialisation parameter controls the degree of parallelism to use for recovery.

The parallel clause can be used with the RECOVER command to parallelise media recovery. It is used to specify the degree or the number of parallel slave processes that will be used. The parallel clause can be used with the recover database, recover tablespace and recover data file commands.

Some examples are:
Recover database parallel (degree d instances default);
Recover tablespace tablespace_name parallel (degree d instances I)
Recover datafile ‘datafile_name’ parallel (degree d);
Recover database parallel (degree default);

The default for degree takes a value equal to twice the number of datafiles being recovered. The default for instances takes the instance-level default vaule specified by the initialisation parameter PARALLEL_DEFAULT_MAX_INSTANCES.

If this was to be done in serial, this command can be used:

Recover database noparallel;

During time of experiment, it was unable to execute the recover command thus it was unable to proceed with this experiement.

8.4 PARALLEL REPLICATION

Parallel Replication

Replication of data is for the purpose of keeping an extra copy of data somewhere just in case there is a need say for security reasons. Oracle provides replication mechanisms allowing the user to maintain copies of the database objects in multiple databases. If there were changes to these tables, the changes would be propagated among these databases over database links. The snapshot (SNP backgroup processes perform the replication process. Parallel propagation can be used to enhance throughput for large volumes of replicated data.

Oracle propagates replicated transactions one at a time in the order in which they are committed in the source database if was performed serially. When parallel propagation for a database link is enabled, Oracle uses multiple parallel slave processes to replicate to the corresponding destination.

The built-in package DBMS_DEFER_SYS is used if parallel replication propagation from the SQL*Plus command line is to be used. The DBMS_DEGER_SYS.SCHEDULE_PUSH is then required to be executed. This is the procedure for the destination database link. The desired degree of parallelism is passed as the value for the parallelism argument.

Below is an example of setting the degree of parallelism for replication propagation using the dBMS_DEFER_SYS. SCHEDULE_PUSH procedure.

```
SQL> execute dbms_defer_sys.schedule_push (-
> destination => 'finprod.world', -
> interval => 'SYSDATE+1/24' -
> NEXT_DATE => 'SYSDATE +1/24', -
PARALLELISM => 6);
```

Due to Oracle access, the dbms_defer_sys.schedule_push was not available thus it was not possible to experiment with this feature.

CHAPTER 9 – CONCLUSION

Although the project was undertaken with some difficulties, it was seen that the majority of the tasks were completed. It was somewhat disappointing not be able to see the timing of each query so that each of the serial and parallel versions can be compared. However, it was possible to see how they differ using the *explain plan* and the *v$pq_tqstat* table stored by ORACLE. Through the *explain plan* feature, it was possible to see how many operations were undertaken and in which order by each of the queries while the *v$pq_tqstat* table allows the viewing of the number of processes used in the parallel execution.

Many different methods of performing the same task was learnt thus this project, although was trial and error; it gave me a hands on feeling of ORACLE.
It was more to do with familiarisation and learning about ORACLE's implementation of parallelism and how it works. Through the seeming less experiments, it was gathered that parallelism should be used for larger sized databases where using serial execution is not feasible because of the time taken to run one query. Using parallelism usually improves the speed but at times will reduce the response time because it is using more than one process and it can be the fact that the tables are far too small for parallelism to perform.

Some parts of the experiments were unable to complete to the end because of the availability of the utilities for ORACLE such as the loader utilities. It runs separate from SQL*plus and because of this, it was impossible to complete that part. Other parts such as getting the timing to work and getting the parallel loading or recovery was also unsuccessful because the service was not available for experimentation.

Most work was put into the experimentation of ways in which parallelism can be used in SQL. Both DML and DDL can be executed in parallel and thus it proves to be a very useful tool when known how to use it.

~ ~ ~

REFERENCES

1. Oracle Manual via Internet
2. BUS5071 – Database Systems and Data Management Lecture notes
3. BUS4410 – Advanced Programming for Database Applications Lecture notes
4. Niemiec, Richard J., Oracle performance tuning tips & techniques / Rich Niemiec., Berkeley : Osborne/McGraw-Hill, c1999.
5. Michael J. Corey … [et al.]., Oracle8 tuning, Berkeley : Osbourne McGraw-Hill, c1998.