Registration Number: IT22178640

Name: Dayana Priyadharshani Kumar

Module: IT3031 - Database Systems and Data Driven Applications

Practical: 07 (Indexes)

--Check what are the tables already you have

SELECT table_name FROM user_tables;

--Tables created in SQL Plus.

--Client table

```
CLN NAME         ADDRESS
--- ------------ ------------------------------
c01 John Smith   3 East Av, Bentley, WA 6102
c02 Jhonson      6 West Av, Westray, NY 6103
```

--Stock Table

```
COMPANY      PRICE   DIVIDEND        EPS
------- ---------- ---------- ----------
BHP           10.5        1.5        3.2
IBM             70       4.25         10
INTEL         76.5          5       12.4
FORD            40          2        8.5
GM              60        2.5        9.2
INFOSYS         45          3        7.8

6 rows selected.
```

--Trading Table

```
COMPANY EXCHANGE
------- ------------
BHP     Sydney
BHP     New York
IBM     New York
IBM     London
IBM     Tokyo
INTEL   New York
INTEL   London
FORD    New York
GM      New York
INFOSYS New York

10 rows selected.
```

--Purchase Table

```
CLN COMPANY PDATE            QTY      PRICE
--- ------- --------- ---------- ----------
c01 BHP     02-OCT-01       1000         12
c01 BHP     08-JUN-02       2000       10.5
c01 IBM     12-FEB-00        500         58
c01 IBM     10-APR-01       1200         65
c01 INFOSYS 11-AUG-01       1000         64
c02 INTEL   30-JAN-00        300         35
c02 INTEL   30-JAN-01        400         54
c02 INTEL   02-OCT-01        200         60
c02 FORD    05-OCT-99        300         40
c02 GM      12-DEC-00        500       55.5

10 rows selected.
```

--Plan Table created

```
SQL> create table PLAN_TABLE (
  2     statement_id    varchar2(30),
  3     timestamp       date,
  4     remarks         varchar2(80),
  5     operation       varchar2(30),
  6     options         varchar2(255),
  7     object_node     varchar2(128),
  8     object_owner    varchar2(30),
  9     object_name     varchar2(30),
 10     object_instance numeric,
 11     object_type     varchar2(30),
 12     optimizer       varchar2(255),
 13     search_columns  number,
 14     id              numeric,
 15     parent_id       numeric,
 16     position        numeric,
 17     cost            numeric,
 18     cardinality     numeric,
 19     bytes           numeric,
 20     other_tag       varchar2(255),
 21     partition_start varchar2(255),
 22          partition_stop  varchar2(255),
 23          partition_id    numeric,
 24     other           long,
 25     distribution    varchar2(30),
 26     cpu_cost        numeric,
 27     io_cost         numeric,
 28     temp_space      numeric);

Table created.
```

Display the client number and name of clients who have made large purchases. A large purchase occurs when a client purchases more than 1000 shares from a company at any given time.

```
SELECT DISTINCT CLIENT.CLNO, CLIENT.NAME

FROM CLIENT

INNER JOIN PURCH ON CLIENT.CLNO = PURCH.CLNO

WHERE PURCH.QTY > 1000;
```

OUTPUT:

```
CLN NAME
--- ------------
c01 John Smith
```

--Session altered commands

--Try to get **all** the rows in the **most efficient** way, even if the first few rows take a little longer.

```
SQL> ALTER SESSION SET OPTIMIZER_MODE = ALL_ROWS;

Session altered.
```

When choosing how to run queries, give **more importance to CPU cost**, not just how much disk I/O is used.

```
SQL> ALTER SESSION SET "_optimizer_cost_model"=CPU;

Session altered.
```

(a) Use EXPLAIN PLAN to find Oracle's generated query plan for the above query by executing the following statement.

```
SQL> explain plan for
  2  select c.clno, c.name
  3  from client c, purch p
  4  where c.clno = p.clno and p.qty > 1000;

Explained.
```

(b) You can execute the script utlxpls.sql to view the query plan and associated costs. Describe each step of the query plan.

```
Plan Table
--------------------------------------------------------------------------------
| Operation and options            |  Object  | cost     | cpu_cost | io_cost  |
--------------------------------------------------------------------------------
| SELECT STATEMENT                 |          | 3        |18063     | 3        |
|  HASH JOIN                       |          | 3        |18063     | 3        |
|   NESTED LOOPS                   |          | 3        |18063     | 3        |
|    NESTED LOOPS                  |          | 3        |18063     | 3        |
|     STATISTICS COLLECTOR         |          | || |
|      TABLE ACCESS FULL           |PURCH     | 2        |9721      | 2        |
|     INDEX UNIQUE SCAN            |SYS_C0082 | 0        |1050      | 0        |
|    TABLE ACCESS BY INDEX ROWID   |CLIENT    | 1        |8341      | 1        |
|   TABLE ACCESS FULL              |CLIENT    | 1        |8341      | 1        |

Plan Table
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

12 rows selected.
```

This query plan shows how Oracle will execute the SQL query, step by step, and provides insights into the operations, costs, and access methods used. Here's a breakdown of the key components in this plan:

1. SELECT STATEMENT

- Operation and options: The main operation for the query, indicating that it's a SELECT query.
- Cost: 3. The overall cost of executing this query.
- CPU Cost: 18063. The estimated CPU resources needed for this query.
- I/O Cost: 3. The estimated I/O resources needed for this query.

2. HASH JOIN

- This is the method used for joining two tables. A Hash Join is used when no indexes are available, or when the optimizer believes it's the most efficient way to join large sets of data.
- The Hash Join indicates that the database will perform a join between the two tables (client and purch) based on the common clno column.

3. Nested Loops (Repeated Twice)

- The Nested Loop Join is used when one of the tables is small and can be scanned for every row in the larger table.
- This operation is nested, meaning that the database will loop through the records of one table and, for each record, it will perform a lookup on the other table.
- This is commonly used for equi-joins (like c.clno = p.clno), where rows from both tables are compared.

4. Statistics Collector

- This step collects execution statistics for the query (e.g., how many rows were processed).

5. TABLE ACCESS FULL for PURCH

- Object: PURCH
- Cost: 2. The cost of a full table scan on the purch table.
- I/O Cost: 9721. The I/O cost of accessing this table.
- This indicates that Oracle is performing a full table scan on the purch table to find all matching records.

6. INDEX UNIQUE SCAN on SYS_C0082

- Object: SYS_C0082 (likely an index on client.clno).
- Cost: 0. The cost for the index scan is negligible.
- CPU Cost: 1050. The CPU cost associated with scanning the index.
- This means Oracle is using an index to efficiently access the client table based on the clno column.

7. TABLE ACCESS BY INDEX ROWID for CLIENT

- Object: CLIENT
- Cost: 1. The cost of retrieving data from the client table using the index rowid.
- I/O Cost: 8341. The I/O cost for accessing the client table.
- This indicates that Oracle is using an index to fetch rows from the client table by accessing the rows identified by the index.

8. TABLE ACCESS FULL for CLIENT

- Object: CLIENT
- Cost: 1. The cost of performing a full table scan on the client table.
- I/O Cost: 8341. The I/O cost for scanning the client table.

(C). Next, create indexes for client and purchase tables. In Oracle you can use the CREATE INDEX statement to create indexes.

```
SQL> CREATE INDEX index_1 ON client(CLNO, NAME);

Index created.
```

```
SQL> CREATE INDEX index_2 ON purch(QTY, CLNO);

Index created.
```

--Re-executed plan table for the same query

```
SQL> explain plan for
  2  select c.clno, c.name
  3  from client c, purch p
  4  where c.clno = p.clno and p.qty > 1000;

Explained.
```

(c) Re-execute the explain plan in part (a). Use utlxpls.sql to view the query plan and associated costs. Explain the query plan and compare it with the query plan in part (b.)

```
Plan Table
--------------------------------------------------------------------------------
| Operation and options            |  Object   | cost     | cpu_cost | io_cost  |
--------------------------------------------------------------------------------
| SELECT STATEMENT                 |           | 2        |14843     | 2        |
|  HASH JOIN                       |           | 2        |14843     | 2        |
|   NESTED LOOPS                   |           | 2        |14843     | 2        |
|    STATISTICS COLLECTOR          |           | || |
|     INDEX RANGE SCAN             |INDEX_2    | 1        |7521      | 1        |
|    INDEX RANGE SCAN              |INDEX_1    | 1        |7321      | 1        |
|   INDEX FULL SCAN                |INDEX_1    | 1        |7321      | 1        |
--------------------------------------------------------------------------------

10 rows selected.
```

Explanation of the query plan

SELECT STATEMENT

- This is the root operation for the SQL query execution.
- Cost: 2
- CPU Cost: 14843
- I/O Cost: 2

HASH JOIN

- The query uses a hash join to combine data from the involved tables.
- Cost: 2
- CPU Cost: 14843
- I/O Cost: 2

Nested Loops

- The join method is nested loops, indicating a row-by-row comparison between tables.

STATISTICS COLLECTOR

- Collecting statistics for the query execution, such as the number of rows processed.

INDEX RANGE SCAN on INDEX_2

- An index range scan is used on INDEX_2, which is likely a range-based lookup on one of the columns.
- Cost: 1
- CPU Cost: 7521
- I/O Cost: 1

INDEX RANGE SCAN on INDEX_1

- Another index range scan is performed on INDEX_1, similar to the previous step.
- Cost: 1
- CPU Cost: 7321
- I/O Cost: 1

INDEX FULL SCAN on INDEX_1

- Index Full Scan indicates that the query is scanning all the rows in INDEX_1 without filtering by a range.
- Cost: 1
- CPU Cost: 7321
- I/O Cost: 1

Comparison of two plans and their features

Hash Join Method:

- Plan 1: Uses a Hash Join for combining tables.
- Plan 2: Also uses a Hash Join, but there's a slight difference in cost and underlying operations.

Nested Loops:

- Plan 1: Uses Nested Loops for the join operation.
- Plan 2: Similarly uses Nested Loops, but the operations involve different indexes.

Statistics Collection:

- Plan 1: The statistics collection operation occurs at a deeper level within the nested loops.
- Plan 2: Statistics collection happens within the nested loops, indicating a similar approach.

Table Access for PURCH:

- Plan 1: Accesses the PURCH table using Full Table Scan.
- Plan 2: Plan 2 doesn't reference the PURCH table at all, suggesting a different underlying data structure or query design.

Table Access for CLIENT:

- Plan 1: Uses Full Table Scan for the CLIENT table.
- Plan 2: Uses Index Scans (both range scan and full scan) for the CLIENT table.

Index Scan Method:

- Plan 1: Uses Index Unique Scan on SYS_C0082.
- Plan 2: Uses Index Range Scan on INDEX_1 and INDEX_2, showcasing a different indexing strategy.

Cost Estimates:

- Plan 1: Has a higher CPU Cost and I/O Cost overall, reflecting a more expensive execution.
- Plan 2: Shows a slightly lower Cost and a more balanced distribution between CPU and I/O Cost.

Object Access:

- Plan 1: Accesses CLIENT using both Index RowID and Full Table Scan.
- Plan 2: Accesses CLIENT using Index Range Scan and Index Full Scan, optimizing the index access.

In summary, Plan 2 is likely the better plan because:

- It uses index range scans and index full scans, which are typically more efficient than full table scans.
- It relies on index-based access for the CLIENT table, making the query more optimized for data retrieval, especially if the CLIENT table is large.
- Plan 1, on the other hand, relies on full table scans for both PURCH and CLIENT tables, which can be slower, particularly for large tables.