CSCI317 – Final Project

Student Name: Trang Nguyen

Student Number: 6166994 – Username: tttn941

Solution 4:

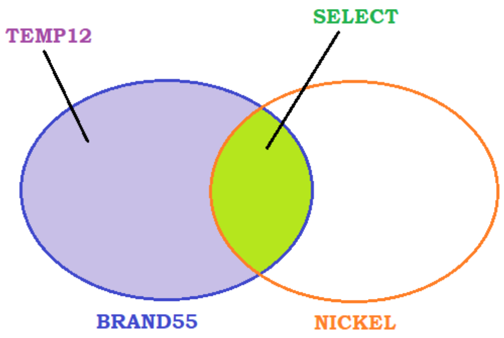
(1) Short description of an improvement

The statements in `task4.java` create three tables, including:

- `BRAND55` containing all parts whose brand is "`Brand#55`"
- `NICKEL` containing all parts of type "`ECONOMY BRUSHED NICKEL`"
- `TEMP12` containing all parts in `BRAND55` but not in `NICKEL` (i.e., a left outer join of the aforementioned tables)

Then, the final `SELECT` statement in `task4.java` fetches the `P_PARTKEY`s that are in `BRAND55` table but not in `TEMP12`, ordered by `P_PARTKEY`.

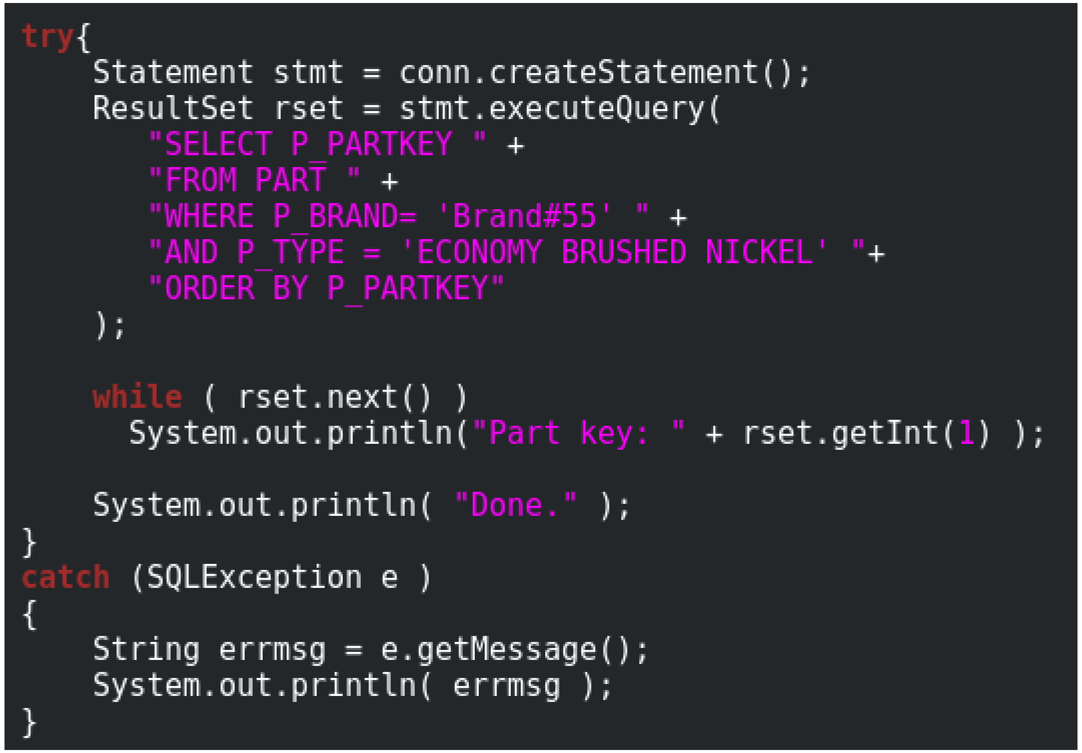
The figure below is a Venn Diagram demonstrating the three tables created and the result set of the `SELECT` statement in `task4.java`.



Based on the Venn Diagram above, what `task4.java` is trying to achieve is to retrieve all rows in `PART` where the brand name is "`Brand#55`" and the type is "`ECONOMY BRUSHED NICKEL`".

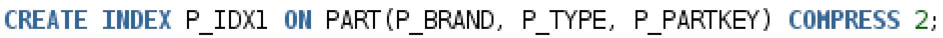
Thus, to improve the JDBC application `task4.java`, `solution4.java` eliminates all creations of the tables and directly retrieves the result set via a single `SELECT` statement.

The simplified `SELECT` statement is as follows:



```
try{
    Statement stmt = conn.createStatement();
    ResultSet rset = stmt.executeQuery(
       "SELECT P_PARTKEY " +
       "FROM PART " +
       "WHERE P_BRAND= 'Brand#55' " +
       "AND P_TYPE = 'ECONOMY BRUSHED NICKEL' "+
       "ORDER BY P_PARTKEY"
    );

    while ( rset.next() )
      System.out.println("Part key: " + rset.getInt(1) );

    System.out.println( "Done." );
}
catch (SQLException e )
{
    String errmsg = e.getMessage();
    System.out.println( errmsg );
}
```

To further improve the `SELECT` statement, a compressed index on `PART(P_BRAND, P_TYPE, P_PARTKEY)` is created server-side so that the query can traverse the index without the need of the table access.

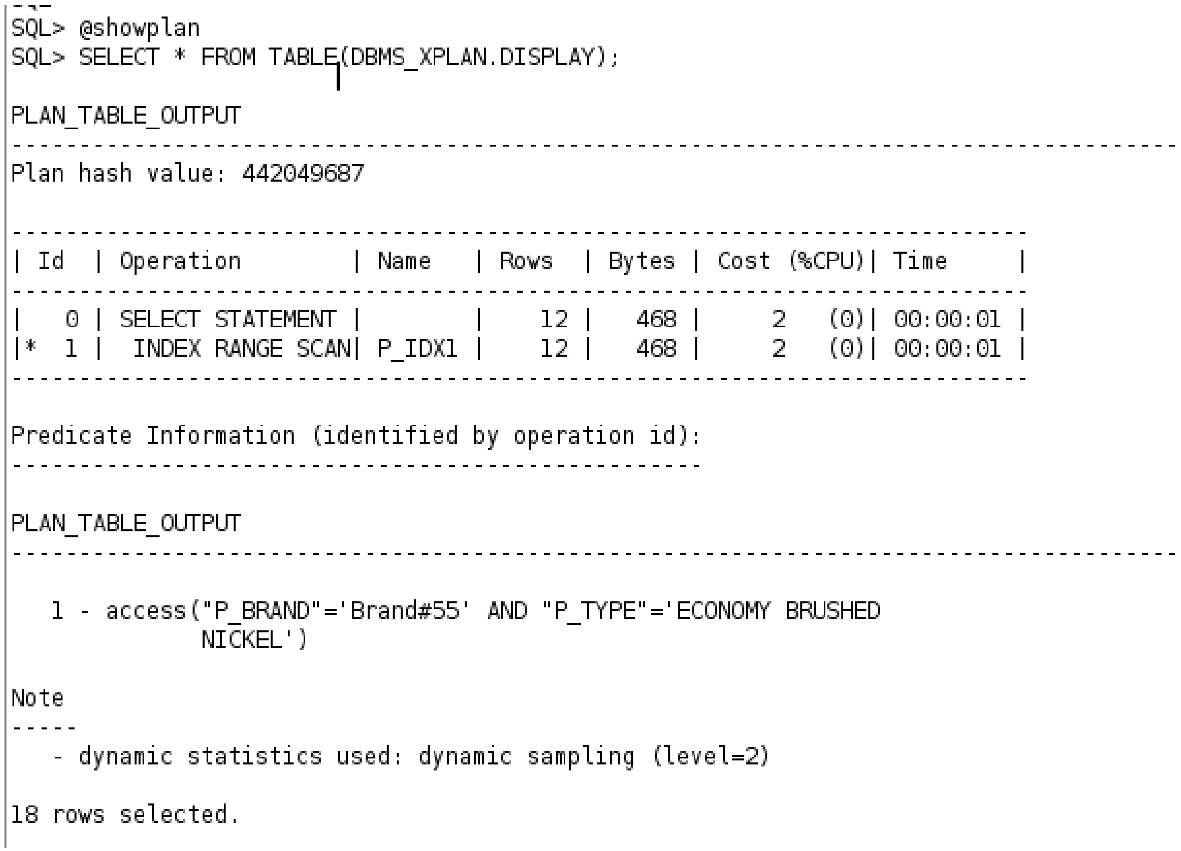


```
CREATE INDEX P_IDX1 ON PART(P_BRAND, P_TYPE, P_PARTKEY) COMPRESS 2;
```

(2) Information about the benefits from an improvement

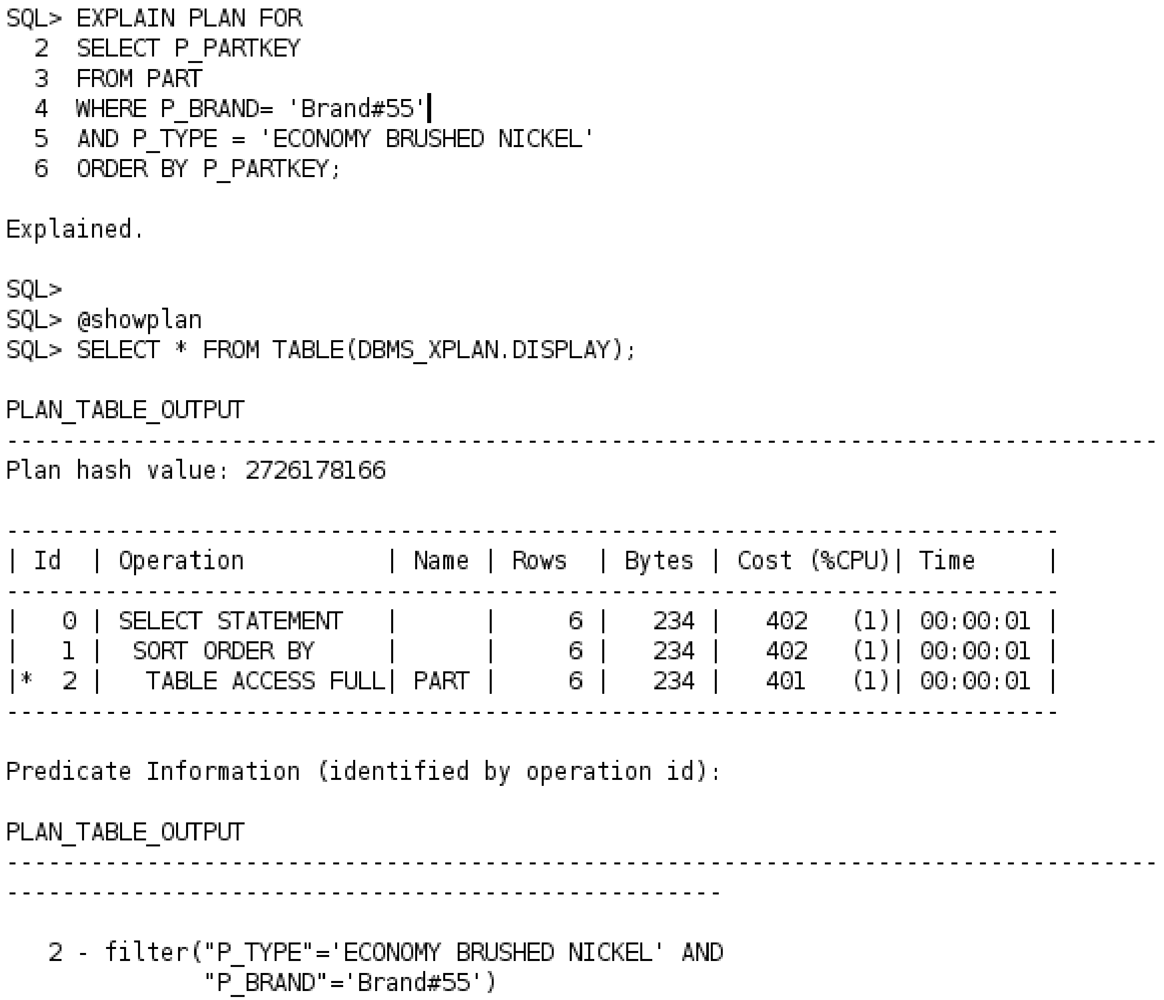
- The single `SELECT` statement eliminates the needs for creating multiple tables, which saves space in the persistent storage.
- Elimination of the table creation also allows us to save unnecessary operations that are time and resource consuming, especially when the operations are performed indirectly through the APIs and not directly on the DBMS.
- Since the eliminated operations read and write to the database, avoiding using them allows us to reduce the chances of the database being corrupted due to any error such as hardware failure or connection error.
- The single `SELECT` statement also makes the code more readable.
- The index created speeds up the `SELECT` statement as it allows for a horizontal traversal of the index without the need to access the table.

The figures below demonstrate the costs of the query with and without the index. Note the difference in the CPU Costs of the operations.



```
SQL> @showplan
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------
Plan hash value: 442049687

---------------------------------------------------------------------------
| Id  | Operation         | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT |        |    12 |   468 |     2   (0)| 00:00:01 |
|*  1 |  INDEX RANGE SCAN| P_IDX1 |    12 |   468 |     2   (0)| 00:00:01 |
---------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------

   1 - access("P_BRAND"='Brand#55' AND "P_TYPE"='ECONOMY BRUSHED
              NICKEL')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)

18 rows selected.
```

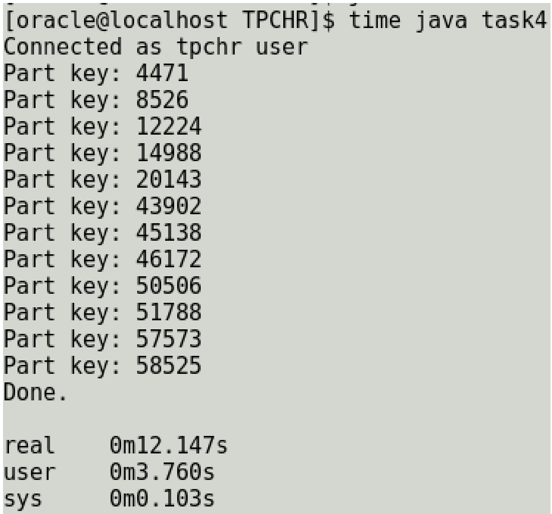
Query processing plan of the SELECT statement *with* index



```
SQL> EXPLAIN PLAN FOR
  2  SELECT P_PARTKEY
  3  FROM PART
  4  WHERE P_BRAND= 'Brand#55'
  5  AND P_TYPE = 'ECONOMY BRUSHED NICKEL'
  6  ORDER BY P_PARTKEY;

Explained.

SQL>
SQL> @showplan
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

PLAN_TABLE_OUTPUT
---------------------------------------------------------------------------------
Plan hash value: 2726178166

---------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     6 |   234 |   402   (1)| 00:00:01 |
|   1 |  SORT ORDER BY     |      |     6 |   234 |   402   (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| PART |     6 |   234 |   401   (1)| 00:00:01 |
---------------------------------------------------------------------------

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT
---------------------------------------------------------------------------------
---------------------------------------------------

   2 - filter("P_TYPE"='ECONOMY BRUSHED NICKEL' AND
              "P_BRAND"='Brand#55')
```

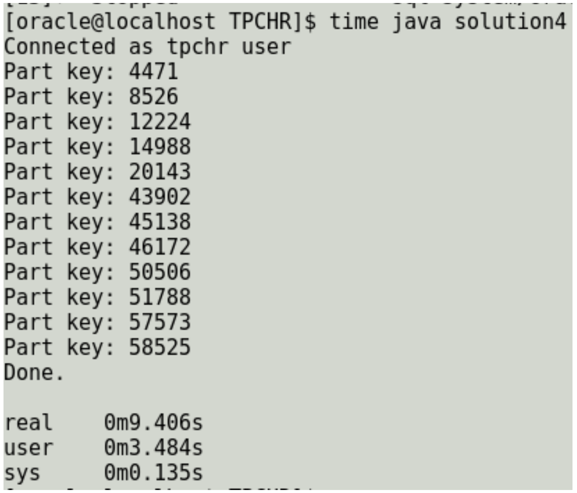
Query processing plan of the SELECT statement *without* index

The figures below demonstrate the run time for each JDBC application:



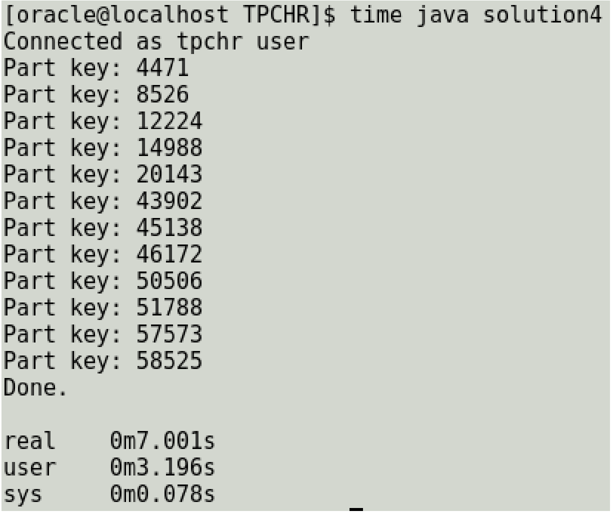
```
[oracle@localhost TPCHR]$ time java task4
Connected as tpchr user
Part key: 4471
Part key: 8526
Part key: 12224
Part key: 14988
Part key: 20143
Part key: 43902
Part key: 45138
Part key: 46172
Part key: 50506
Part key: 51788
Part key: 57573
Part key: 58525
Done.

real    0m12.147s
user    0m3.760s
sys     0m0.103s
```

task4.java run time



```
[oracle@localhost TPCHR]$ time java solution4
Connected as tpchr user
Part key: 4471
Part key: 8526
Part key: 12224
Part key: 14988
Part key: 20143
Part key: 43902
Part key: 45138
Part key: 46172
Part key: 50506
Part key: 51788
Part key: 57573
Part key: 58525
Done.

real    0m9.406s
user    0m3.484s
sys     0m0.135s
```

solution4.java run time without the index



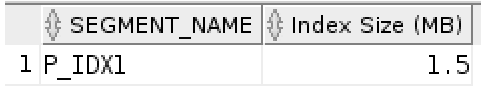
```
[oracle@localhost TPCHR]$ time java solution4
Connected as tpchr user
Part key: 4471
Part key: 8526
Part key: 12224
Part key: 14988
Part key: 20143
Part key: 43902
Part key: 45138
Part key: 46172
Part key: 50506
Part key: 51788
Part key: 57573
Part key: 58525
Done.

real    0m7.001s
user    0m3.196s
sys     0m0.078s
```

Solution4.java run time with the index

(3) information about the costs of an improvement (i.e., documented investments into transient and persistent storage)

In terms of transient storage, there is an overhead in creating a Statement and in executing it for the first time. However, it’s not significant to the overall application's performance. No new investment into transient storage has been made compared to task4.java.

However, an investment of 1.5MB into persistent storage is needed for the index on `PART(P_BRAND, P_TYPE, P_PARTKEY)` to speed up the query.



	SEGMENT_NAME	Index Size (MB)
1	P_IDX1	1.5

(4) a report from implementation of an improvement.

```
/* IMPROVED JDBC APPLICATION – solution4.java */
import java.sql.*;
class solution4
{
  public static void main (String args [])
       throws SQLException, ClassNotFoundException
  {
    // Load the Oracle JDBC driver
    Class.forName ("oracle.jdbc.driver.OracleDriver");
    Connection conn = DriverManager.getConnection
       ("jdbc:oracle:thin:@localhost:1521:db",  "tpchr", "oracle");
      System.out.println( "Connected as tpchr user");
    try{
        Statement stmt = conn.createStatement();
        ResultSet rset = stmt.executeQuery(
           "SELECT P_PARTKEY " +
           "FROM PART " +
           "WHERE P_BRAND= 'Brand#55' " +
           "AND P_TYPE = 'ECONOMY BRUSHED NICKEL' "+
           "ORDER BY P_PARTKEY"
      );
        while ( rset.next() )
          System.out.println("Part key: " + rset.getInt(1) );
        System.out.println( "Done." );
    }
    catch (SQLException e ) {
      String errmsg = e.getMessage();
      System.out.println( errmsg );
    }
  }
}
```

/* INDEX TO IMPROVE THE PERFORMANCE OF THE QUERY */

```
SQL> /* Create the index */
SQL> CREATE INDEX P_IDX1 ON PART(P_BRAND, P_TYPE, P_PARTKEY) COMPRESS 2;

Index P_IDX1 created.

SQL>
SQL> EXPLAIN PLAN FOR
  2  SELECT P_PARTKEY
  3  FROM PART
  4  WHERE P_BRAND= 'Brand#55'
  5  AND P_TYPE = 'ECONOMY BRUSHED NICKEL'
  6  ORDER BY P_PARTKEY;

Explained.
```

```
SQL> @showplan
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

PLAN_TABLE_OUTPUT
---------------------------------------------------------------------------------------------------------
Plan hash value: 442049687

---------------------------------------------------------------------------
| Id  | Operation         | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT |        |    12 |   468 |     2   (0)| 00:00:01 |
|*  1 |  INDEX RANGE SCAN| P_IDX1 |    12 |   468 |     2   (0)| 00:00:01 |
---------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

PLAN_TABLE_OUTPUT
---------------------------------------------------------------------------------------------------------
   1 - access("P_BRAND"='Brand#55' AND "P_TYPE"='ECONOMY BRUSHED
              NICKEL')
```

```
Note
-----
   - dynamic statistics used: dynamic sampling (level=2)

18 rows selected.
SQL> /* Find the size of the index created above */
SQL> select segment_name,
  2      sum(bytes)/1024/1024 as "Index Size (MB)"
  3  from user_segments
  4  where segment_name='P_IDX1'
  5  group by segment_name;

SEGMENT_NAME    Index Size (MB)
------------    ---------------
P_IDX1                     1.5

1 rows selected.
```

```
SQL> /* Drop the index */

SQL> DROP INDEX P_IDX1;


Index P_IDX1 dropped.
```