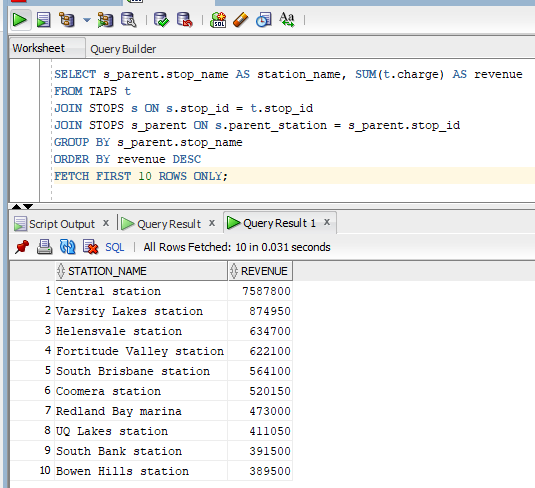
Name: Nguyen Van Quoc Chuong

Id: 47787810

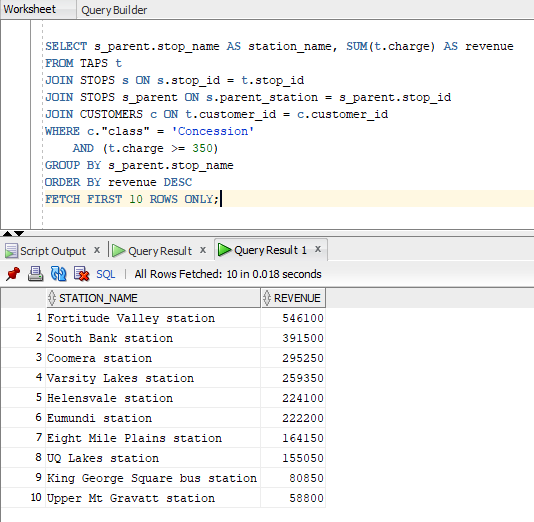
INFS2200 Assignment 2 report

**TASK 1: VIEWS**

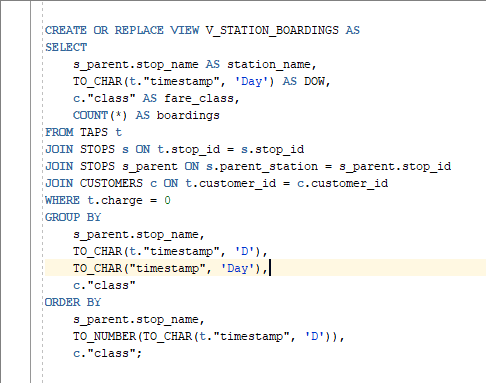
Task 1.1:

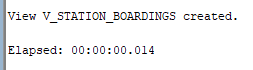


Task 1.2:

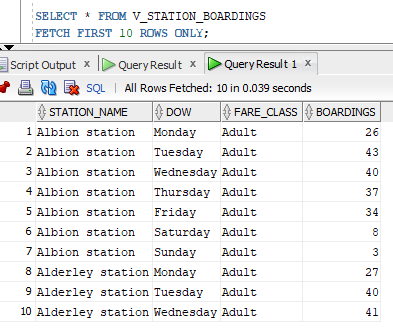


Task 1.3:

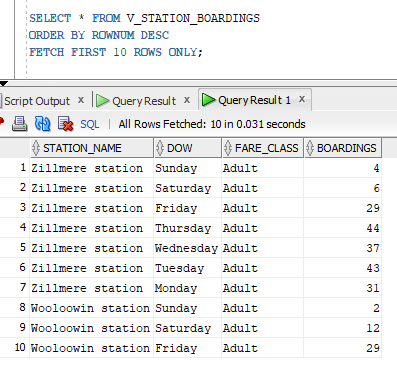




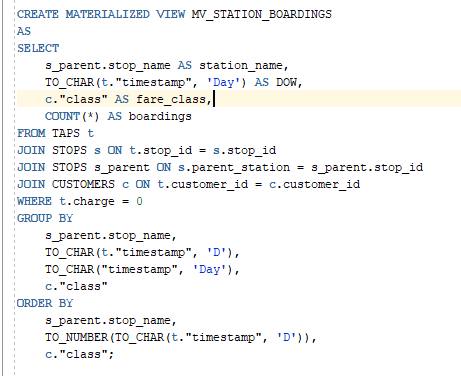
Top 10 stations



Last 10 stations



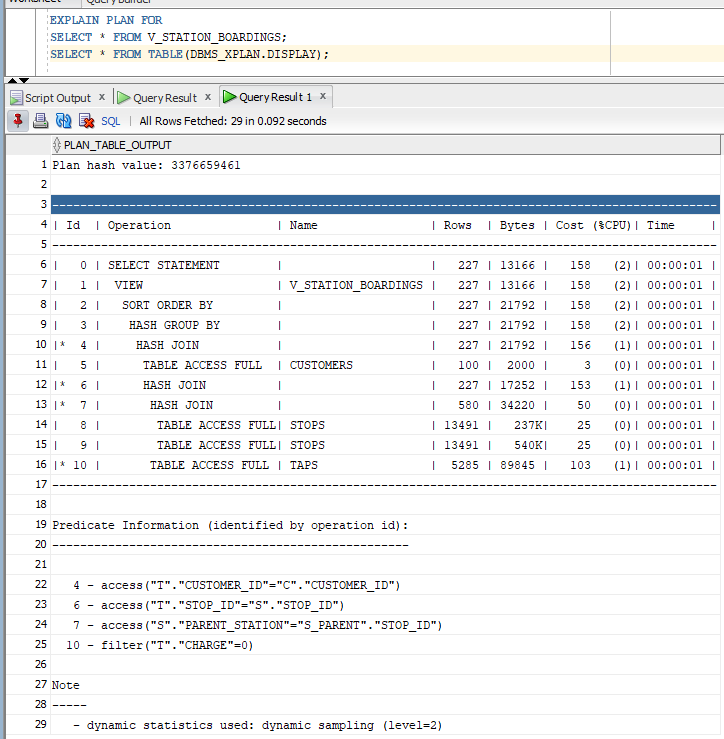
Task 1.4:





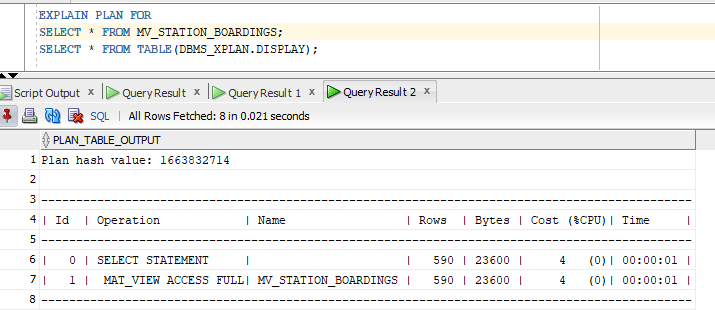
Task 1.5:

V\_STATION\_BOARDINGS:





MV\_STATION\_BOARDINGS:



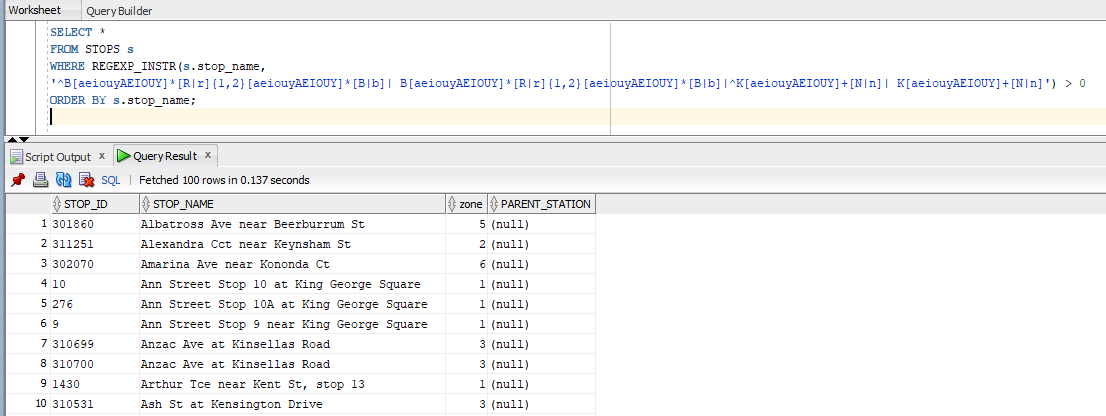


The cost for this query is significantly lower compared to the previous query you provided. In this plan, the cost is only 4, while in the previous plan, it was 158. This indicates that accessing the data from the Materialized View is much more efficient in terms of resource usage and performance. The Elapsed time is also lower a bit.

-------------------------------------------------------------------------------------------------------------------------------------

**TASK 2: Indexes**

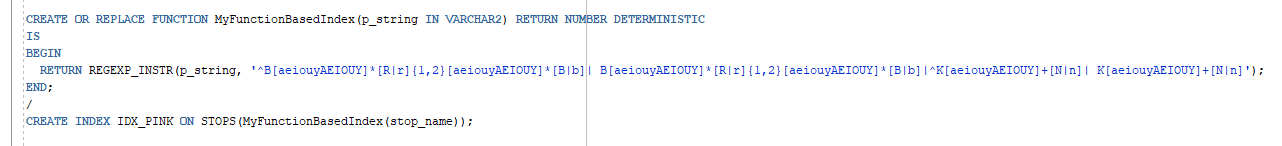
Task 2.1:

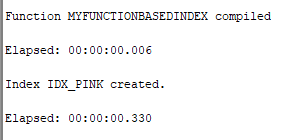


Last 10 rows



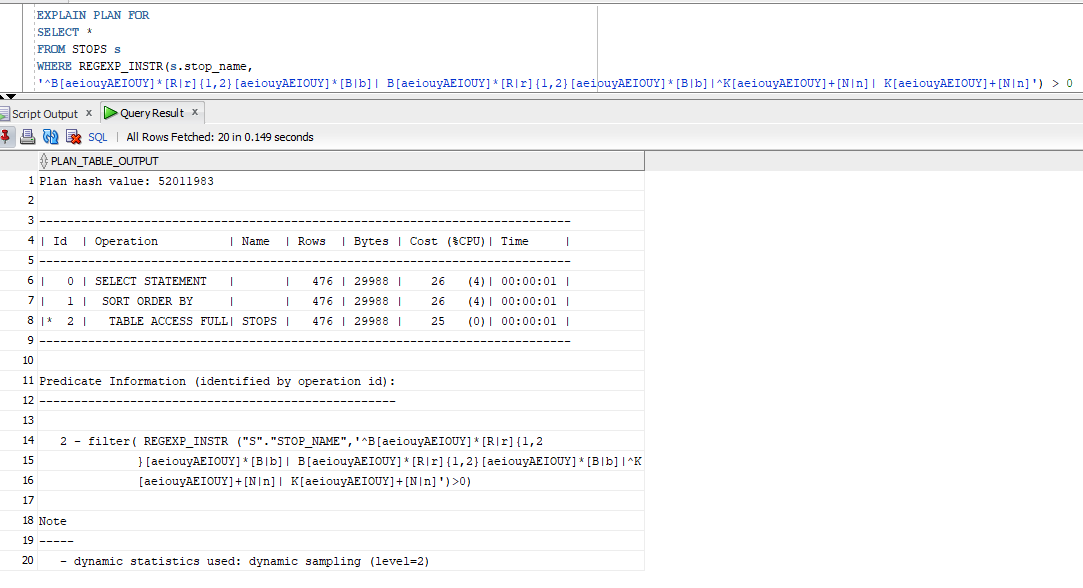
Task 2.2:





Task 2.3:

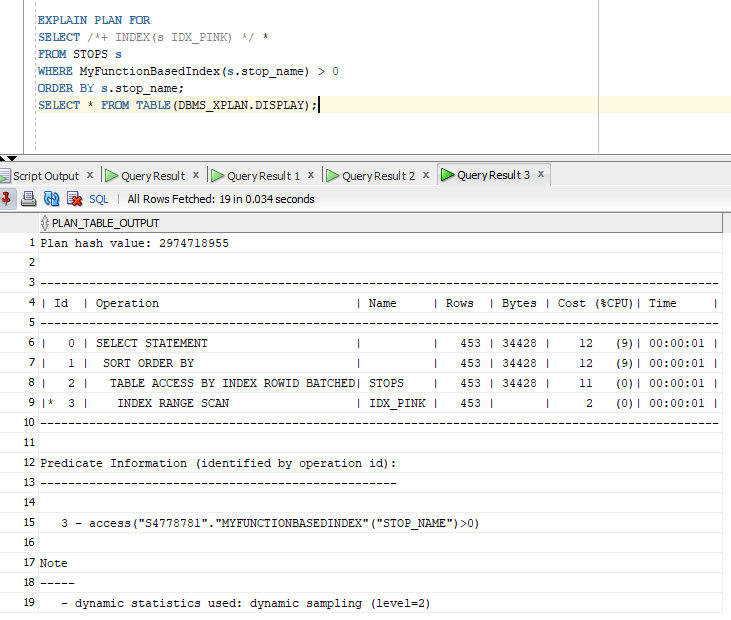
Execution plan WITHOUT index:





* The row estimate difference between the SELECT statement (453) and the explain plan (476) is attributed to Oracle's dynamic sampling, which refines cost estimates. The actual number of rows returned by the query (453) is the accurate figure to consider. The estimate in the explain plan (476) is an intermediate step and may not precisely match the actual result.

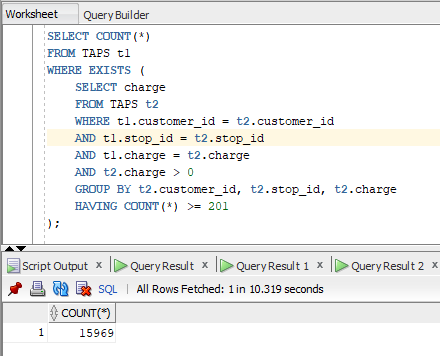
Execution plan WITH index:



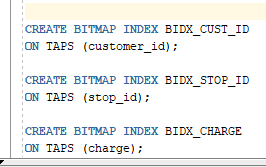


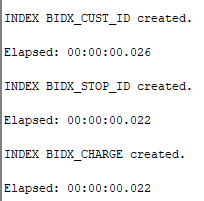
* We can see that, with index, the cost was reduced to 12, but the CPU% was 9%. The Elapsed time and Bytes with index are higher. Generally, the first query without the function-based index perform a full table scan, while the second query, which uses a function-based index, is much more efficient in terms of cost, as it utilizes the index to perform the regular expression matching. This can significantly improve query performance when dealing with large datasets, as it avoids scanning the entire table.

Task 2.4:



Task 2.5:

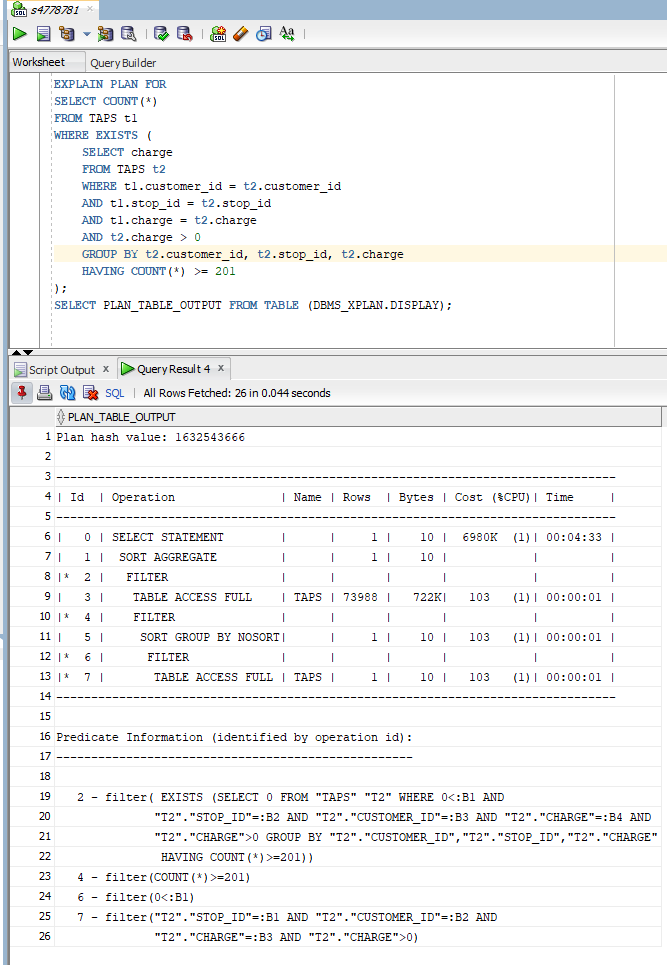




Task 2.6:

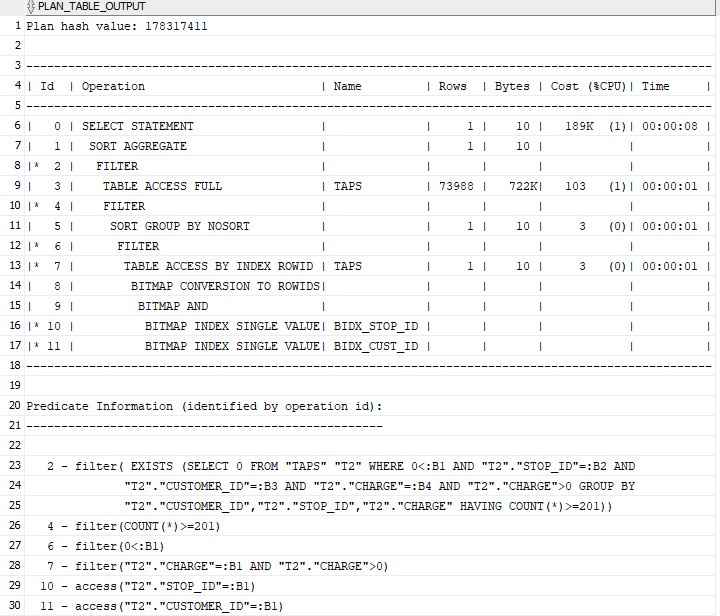
Before using Bitmap Index:





After using Bitmap Index:

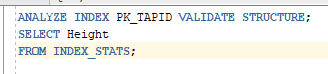




The bitmap index greatly enhances efficiency. In the two scenarios, the SELECT statement results in a cost of 6980K without the bitmap index and only 189K with it. Since the elapsed time varies, I rely solely on the consistent cost metric to evaluate its effectiveness.

-------------------------------------------------------------------------------------------------------------------------------------

**TASK 3: EXECUTION PLANS**

Task 3.1.a:  

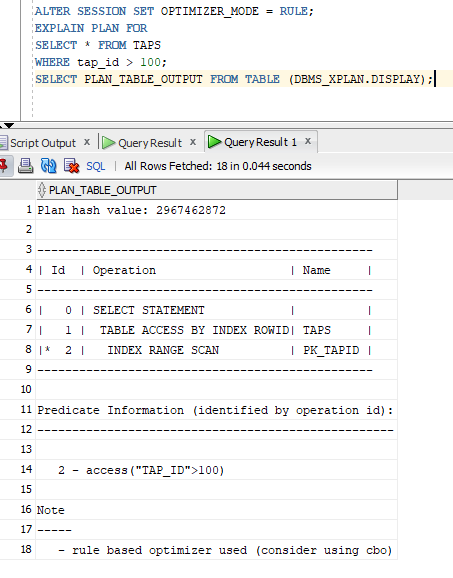
The height of B+ tree is 2.

Task 3.1.b:  

The number of leaf blocks is 138.

Task 3.1.c:  

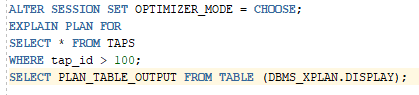
Task 3.2:



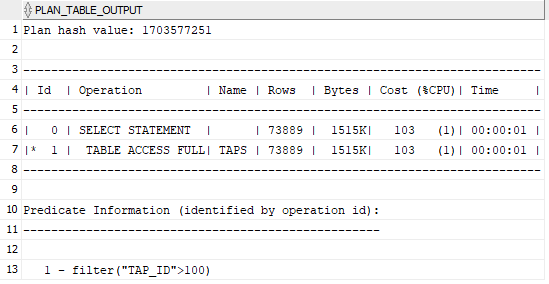
Analyzing the query execution:

* INDEX (RANGE SCAN): The PK\_TAPID index is employed to scan through a range of values, specifically addressing the conditions specified in the WHERE clause.
* TABLE ACCESS BY INDEX ROWID: Rows are retrieved using the index, and it's important to note that a full table scan, as seen in the previous example, is avoided.
* The SELECT statement processes and returns rows that meet the criteria outlined in the WHERE clause.

Task 3.3:





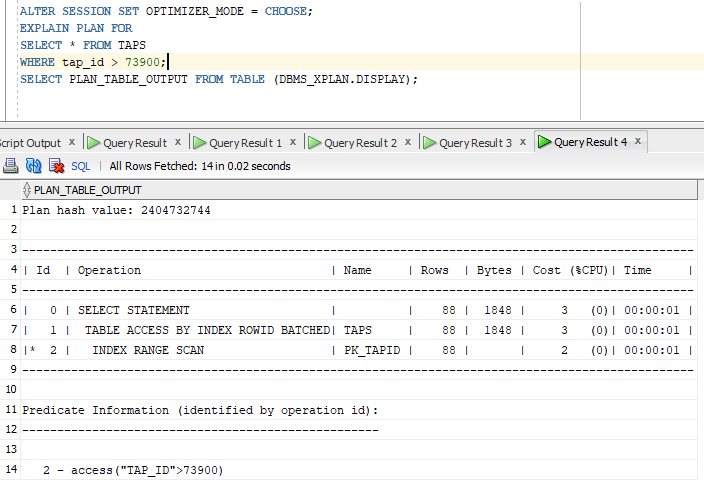


Query processing steps:

* The query scans the entire "TAPS" table.
* For each row, it checks the value in the "TAP\_ID" column to see if it's greater than 100.
* Rows that meet this condition are included in the result set.

Key Difference: The main difference is in the access method. The 3.2 plan used an index range scan, while the 3.3 plan uses a full table scan. The choice of access method can significantly impact query performance, where an index scan is more selective and efficient than a full table scan.

Task 3.4:

Query Processing Steps:

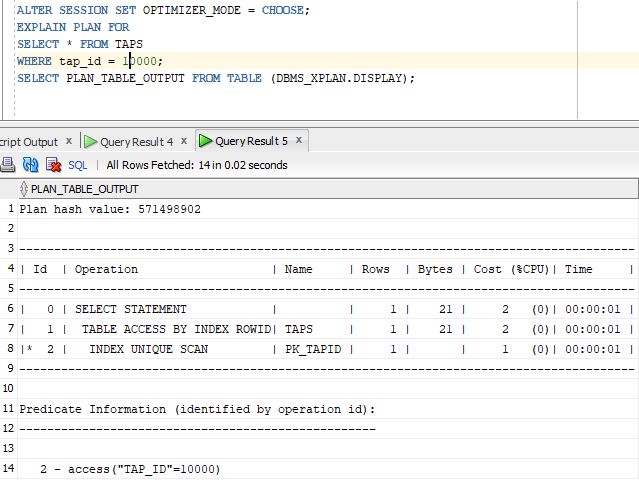
* Initially, an INDEX (UNIQUE SCAN) is performed on the "PK\_TAPID" index, uniquely retrieving specific rows that meet the WHERE clause conditions.
* Subsequently, a TABLE ACCESS BY INDEX ROWID BATCHED operation locates rows using the index efficiently.
* Finally, the SELECT statement delivers the rows that satisfy the WHERE clause criteria.

Key Difference:

* Task 3.3, where "tap\_id" needs to be greater than 100, the condition is not very selective. Oracle's query optimizer opts for a Table Full Scan.
* Conversely, in Task 3.4, the condition is tap\_id > 73900, which is highly selective. Oracle's query optimizer recognizes this and chooses the more efficient Index Range Scan. It starts at the specified range point (73900) and scans through the index, retrieving rows that match the condition within the range.

Task 3.5:





Query Processing Steps:

* Firstly, an INDEX (UNIQUE SCAN) is executed, utilizing the "PK\_TAPID" index for evaluating the WHERE clause condition. This results in the retrieval of a single row identifier from the index.
* Subsequently, a TABLE ACCESS BY INDEX ROWID operation is performed to locate rows using the index (Hashing), avoiding a full table scan.
* Finally, the SELECT statement returns rows that meet the WHERE clause conditions.

Key difference:

* 3.3 plan scans full table while 3.5 using unique scan (or hashing index) for executing operation “=”. That’s why the number of bytes and rows are much smaller than 3.3.