Lab-2 Solutions

```sql
SQL> edit
Wrote file afiedt.buf
1. EXPLAIN PLAN
2. SET STATEMENT_ID = 'HW2-B1'
3. FOR
4. SELECT DIRECTOR
5. FROM MOVIE
6. WHERE VOTES > 4 AND VOTES < 16
SQL> /
Explained.
SQL> SELECT * FROM (DBMS_XPLAN.DISPLAY('PLAN_TABLE', 'HW2-B1'));
PLAN_TABLE_OUTPUT

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (NCPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td>MOVIE</td>
<td>15</td>
<td>315</td>
<td>16 (0)</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID BATCHED</td>
<td>MOVIE</td>
<td>15</td>
<td>315</td>
<td>16 (0)</td>
</tr>
<tr>
<td>2</td>
<td>INDEX RANGE SCAN</td>
<td>MOVIE_VOTES</td>
<td>15</td>
<td>2</td>
<td>0 (0)</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT

2. access('VOTES'>4 AND 'VOTES'<16)

Note
- 'PLAN_TABLE' is old version
16 rows selected.
```

Query 1 has very low selectivity (~.0015), whereas Query 2 has high selectivity (~.53). Index are useful for range queries when selectivity is very low. Following relation on selectivity further supports this argument.

Full Scan Cost = \( \frac{Nr \cdot \text{record size}}{\text{block size}} \)

Index Scan Cost = depth of 2\(^{nd}\)ary index + \( \frac{NDV(\text{votes}) \cdot (\text{key-value size} + \text{pointer value size}) \cdot \text{selectivity}}{\text{block size}} \) + (Number of records \times \text{selectivity})
1 EXPLAIN PLAN
2 SET STATEMENT_ID = 'HW2-C1'
3 FOR
4 SELECT TITLE
5 FROM MOVIE
6# WHERE VOTES < 2000
SQL> /

Explained.

SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY('PLAN_TABLE', 'HW2-C1'));

PLAN_TABLE_OUTPUT

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (MCPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td>MOVIE</td>
<td>4151</td>
<td>95473</td>
<td>23 (0)</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS FULL</td>
<td>MOVIE</td>
<td>4151</td>
<td>95473</td>
<td>23 (0)</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT

1 - filter("VOTES"<2000)

1 EXPLAIN PLAN
2 SET STATEMENT_ID = 'HW2-C2'
3 FOR
4 SELECT TITLE
5 FROM MOVIE
6# WHERE VOTES > 50000 AND VOTES < 100000
SQL> /

Explained.

SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY('PLAN_TABLE', 'HW2-C2'))
2 ;

PLAN_TABLE_OUTPUT

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (MCPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td>MOVIE</td>
<td>009</td>
<td>20907</td>
<td>23 (0)</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS FULL</td>
<td>MOVIE</td>
<td>009</td>
<td>20907</td>
<td>23 (0)</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT

1 - filter("VOTES">50000 AND "VOTES"<100000)

1 EXPLAIN PLAN
2 SET STATEMENT_ID = 'HW2-C3'
3 FOR
4 SELECT TITLE
5 FROM MOVIE
6# WHERE YR > 1980 AND YR < 1950
SQL> /

Explained.

SQL> edit
Wrote file afiled.buf

1 SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY('PLAN_TABLE', 'HW2-C3'))
SQL> /

PLAN_TABLE_OUTPUT

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (MCPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td>MOVIE</td>
<td>3633</td>
<td>79926</td>
<td>23 (0)</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS FULL</td>
<td>MOVIE</td>
<td>3633</td>
<td>79926</td>
<td>23 (0)</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT

1 - filter("YR"<1980 AND "YR">1950)

1 EXPLAIN PLAN
2 SET STATEMENT_ID = 'HW2-C4'
3 FOR
4 SELECT TITLE
5 FROM MOVIE
6# WHERE YR > 1950 AND YR < 1955
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY('PLAN_TABLE', 'HW2-C4'));

PLAN_TABLE_OUTPUT

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (MCPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td>MOVIE</td>
<td>585</td>
<td>11110</td>
<td>12 (0)</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>MOVIE</td>
<td>585</td>
<td>11110</td>
<td>12 (0)</td>
</tr>
<tr>
<td>2</td>
<td>INDEX RANGE SCAN</td>
<td>MOVIE_YR</td>
<td>585</td>
<td></td>
<td>3 (0)</td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT

2 - access("YR">1950 AND "YR"<1955)
<table>
<thead>
<tr>
<th>Query</th>
<th>Selectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>~.42</td>
</tr>
<tr>
<td>Q2</td>
<td>~.091</td>
</tr>
<tr>
<td>Q3</td>
<td>~.37</td>
</tr>
<tr>
<td>Q4</td>
<td>~.051</td>
</tr>
</tbody>
</table>

**Query-4** has the lowest Selectivity

No, the index on votes is not used on Q1 and Q2, because, the selectivity for both Q1 and Q2 is higher than the maximum selectivity at which index search would have been more efficient w.r.t the 2ndary index on votes.

Index on YR is used for Q3 and not used for Q4. It is because, the selectivity for Q3 is high, thus using file scan is more optimal whereas, the selectivity for Q4 is low enough that using index scan is more optimal.

---

**Ans.** The execution plan for Q1 uses INDEX RANGE SCAN, whereas, the execution plan for Q2 uses INDEX FAST FULL SCAN.

**Reason:** In Q1 all the names starting with M refers to sequential search. An index range scan is used when the SQL statement contains a restrictive clause that requires a sequential range of values that are indexes for the table [1]. In Q2 the names ending with m is non-sequential search, however, since we have 2ndary index on name, it is sufficient to search the index to retrieve the necessary results.
Ans. The query plan for Q3 uses FULL TABLE ACCESS operation and plans to retrieve 189 rows, whereas, the query plan for Q4 uses INDEX FAST FULL SCAN and plans to retrieve 944 rows.

Reason: Since, Q3 uses substring function, the index on name is not considered and function is evaluated on the column as is. Q4 uses, inbuilt pattern matching operator, that can work on indexes in Oracle. For function to work on index, a functional index is required.
Ans. Yes, there is significant difference in query plans for Q1 and Q2. Query Q1 is more efficient to execute.

Reason. In query Q1 optimizer first computes the value of MIN operation on the table and generates the single value result. This result then, can be used in the index to compute the final result. Thus, in Q1 the table is only accessed to retrieve the value of DIRECTOR column. In Q2, two copies of table MOVIE is created, and anti-sort merge join is applied on them to generate the desired output. This is very expensive because 1) two copies of the table is created, 2) the tables are sorted on same column, finally they are merged to generate the required result.
Ans. Yes. Q1 is has more optimal execution plan compared to Q2.
Reason. In Q1, nested query is first executed using INDEX RANGE SCAN on title index. Then, INDEX RANGE SCAN on votes index is executed to retrieve all the necessary rows, and title is retrieved using ROWIDs. In Q2, point query on “Kingpin” is first done using INDEX RANGE SCAN. Then, FILTER operation is done using nested loop that requires single scan on m1 copy of MOVIE table with single row of m2. Following this nested loop is performed on the filtered output to retrieve all the TITLE values. Clearly, Q2 is expensive.
Ans. Yes, Q1, Q2 and Q3 differ in their execution plan.

**Access** means we are using something to “access” the data. We only access relevant data. **Filter** means we are getting more data than we need, and we filter it after we get it.

**Reason.** In Q1, Filter is applied on “Casting” table, then Statistics Collector is run to check which works better (NESTED LOOPS or HASH JOIN). Finally, Optimizer chooses HASH JOIN to join the tables (marked by *). NESTED LOOPS are ignored. The total cost of Q1 is $296(23 + 23 + 68 + 91 + 91)$. In Q2, we redirect the optimizer to use INDEX MERGE SORT, that proves to be costly than Q1. The cost for Q2 is 373. In Q3, we redirect the optimizer to use Nested Loop (NL) that proves to be very costly. The cost for Q3 is 28965. Hence, Q1 is the most optimal.