Exercise 2 solution

1. Discuss which query optimisation approach(es) (enumerating all plans, heuristic based, adaptive plans) can be suitable for the following scenarios:

   - **Scenario A**: Given a table with 1 million tuples, run the following query:

     ```
     SELECT customer
     FROM Table
     WHERE spend BETWEEN 100 AND 200
     AND birth_year> 2000;
     ```

   - **Scenario B**: Given 5 tables with 1 million tuples in each table, run a query:

     ```
     SELECT T1.name, T2.salary, T3.qualification, T4.phone, T5.leader
     FROM Table1 T1
         INNER JOIN Table2 T2 ON T2.id = T1.id
         INNER JOIN Table3 T3 ON T3.id = T1.id
         INNER JOIN Table4 T4 ON T4.id = T1.id
         INNER JOIN Table5 T5 ON T5.department = T1.department
     WHERE T1.age > 50;
     ```

   Solution:

   Queries are generally converted to Relational Algebra expressions internally first. Then the system tries to create alternate plans and pick the best plan to execute in terms of execution time. There are two general approaches for this. One is searching/enumerating all the plans and choose the best one. Another approach is using heuristics to choose a plan (or a combination of two can be done as well). Adaptive plan is executing a part/some parts of a query plan first to re-evaluate the cost of the other parts of the query plan that haven't been executed yet, to have a better overall cost estimation (and hence, choosing a better plan).

   For Scenario A, the heuristic approach can be suitable due to the simplicity of the query and small size of the table. For Scenario B, the enumerating approach can be more suitable due to the complexity of the query.

   Adaptive plan is used for better estimation of cost, hence, cannot be used when the query optimiser is purely heuristic based for a query (so cannot be used for scenario A if the plan is heuristic based). Adaptive plan can be used for cost-based query optimiser (either exhaustive enumeration of all plans or a combination), hence, can be used for Scenario B.

2. A particular query on a table A used to run quite efficiently in a DBMS. After inserting many records and deleting many other records from table A, that same query is now taking more time to run, even when the total number of records has not changed. What can be the reason for that? What can you do as the user/database administrator of that DBMS to improve the performance of this query?

Solution:

After insertions and deletions, the statistics of a table may not be updated instantly. As the statistics are used to estimate the cost of a query, wrong statistics are causing wrong cost estimations, and hence the query optimiser is now choosing a query plan that is no longer optimal for that query.

Enforcing statistical recompilation option can be used to update the statistics of the table.

3. When a database needs to join two tables using a page-oriented nested loop join algorithm, both tables are found to be in the main memory. Will it change how the cost of a cost-based optimiser calculates the cost of the query plan?

   Solution: No, the cost calculation of a query plan will be same as if both tables were on disk.

4. What are the possible approaches that you can take to improve query performance in practice? **Solution**:

   1. Use index if necessary
   2. Force a query plan instead of the plan chosen by the optimizer
   3. Enforce statistic recompilation
   4. Rewrite query with parameters for execution plan reuse
   5. Store derived data (when you frequently need derived values and data do not change frequently)
   6. Use pre-joined tables (when tables need to be joined frequently).

5. In the worst case, if there is enough memory only to hold one page/block of each table, what are the estimated cost of simple nested loop join and block nested loop join? What are the number of seeks for simple nested loop join and block nested loop join?

   Solution:

   For simple nested loop join, the estimated cost is $b_r + (n_r \times b_s)$ (number of page access), and $n_r + b_r$ seeks. Note: Regarding seek, b_r is the number of blocks in r relation. The r relation is accessed as part of the outer loop, and eventually all blocks of r will be accessed one by one (by the outer loop), hence there are corresponding b_r seeks. For each tuple of r relation, a block of the s relation needs to be accessed (so, 1 seek for each tuple of r relation). So, for total n_r tuples of r relation, there will be n_r number of seeks. Hence, total seek count is $n_r + b_r$.

   For block nested loop join, the estimated cost is $b_r + (b_r \times b_s)$ (number of page access), and $2 \times b_r$ seeks. Note: Regarding seek, b_r is the number of blocks in r relation. The r relation is accessed as part of the outer loop, and eventually all blocks of r will be accessed one by one (by the outer loop), hence there are corresponding b_r seeks. For each block of r relation, a block of the s relation needs to be accessed (so, 1 seek for each block of r relation). So, for total b_r blocks of r relation, there will be b_r number of seeks. Hence, total seek count is $b_r + b_r = 2 \times b_r$.