Data Systems

Group Assignment 4: Query Processing and Optimization

Sayar and Priyank

[A]

Issues: Even though the queries are supposedly executed simultaneously, this may not be practically possible because of main memory limitations. Every query requires some data blocks to be in main memory (either it is loaded onto main memory or it is already present as a result of a previous operation).

In the process of executing all queries, we might end up doing the same sub-operation multiple times if we execute the queries independently of each other. For example, 'SELECT Fname from EMPLOYEE WHERE Salary > 32000' could easily use the intermediate result from 'SELECT Fname from EMPLOYEE WHERE Salary > 20000'. Thus, scheduling the queries becomes a task of utmost importance. The goal here is to reduce loads and unloads involving main memory. We also try to ensure that pre-computed results are reused for queries executed down the line. In that, we try to access the same relation as less number of times as possible. A set of heuristics and strategies can be used here.

We could look at all the queries beforehand as a collective batch and then process them accordingly. For example, change the order of execution, cache or save intermediate results which can be reused, build temporary indices on certain fields within relations and on fields in relations created as a result of an intermediate join. We can pre-compute the cost of joins as well and pre-compute a few joins needed for later queries. Some common pre-processing for crosses can also be done. Other statistics can be maintained. The optimization would collectively involve: reordering, saving some essential statistics, saving temporary indices after figuring out the most optimal index patterns, and saving necessary intermediate outputs during the actual execution.

[B]

Q1:

1. SELECT Pname FROM PROJECT
2. SELECT Dnum FROM PROJECT

Q2:

1. SELECT Pnumber FROM PROJECT
2. SELECT Plocation FROM PROJECT
3. SELECT Dnumber FROM DEPT_LOCATIONS
4. SELECT Dlocation FROM DEPT_LOCATIONS

5. SELECT Dnumber FROM DEPARTMENT

Q3:

1. SELECT Ssn FROM EMPLOYEE
2. SELECT Fname, Lname FROM EMPLOYEE
3. SELECT Fname FROM EMPLOYEE
4. SELECT Lname FROM EMPLOYEE
5. SELECT Fname, Lname FROM EMPLOYEE WHERE Dno=4
6. SELECT Fname, Lname FROM EMPLOYEE WHERE Dno=1
7. SELECT Dname FROM DEPARTMENT
8. SELECT Pname FROM PROJECT WHERE Dnum=5
9. SELECT Essn FROM DEPENDENT WHERE Sex='M'
10. SELECT Sex FROM DEPENDENT

[C]

Part 1:

On Set 1:

- Π_{Pname}, Π_{Dnum} operating on relation PROJECT

On Set 2:

- $\Pi_{Pnumber}$, $\Pi_{Plocation}$ operating on relation PROJECT
- $\Pi_{Dnumber}$, $\Pi_{Dlocation}$ operating on relation DEPT_LOCATIONS
- $\Pi_{Dnumber}$ operating on relation DEPARTMENT

On Set 3:

- Π_{Ssn}, $\Pi_{(Fname,\ Name)}$, Π_{Fname}, Π_{Lname}, $\Pi_{Dno=4}$, $\Pi_{Dno=1}$ on relation EMPLOYEE
- Π_{Dname} on relation DEPARTMENT
- Π_{Pname}, $\Pi_{Dnum=5}$ on relation PROJECT
- Π_{Essn}, Π_{Sex}, $\Pi_{Sex='M'}$ on relation DEPENDENT

Part 2:

Set 1:

- Get outputs for queries 1 and 2 with one relation access.

Set 2:

- Op1: Get outputs for queries 1 and 2 with one relation access.
- Op2: Get outputs for queries 3 and 4 with one relation access.

- Op3: Get output for query 5

Also, Op1, Op2 and Op3 can be run in parallel using multiprocessing as they are independent of each other.

Set 3:

- Op1: Get outputs for queries 1 thru 6 with one relation access
- Op2: Get output for query 7
- Op3: Get output for query 8
- Op2: Get outputs for queries 9 and 10 with one relation access

Op1, Op2, Op3 and Op4 can be run in parallel using multiprocessing as they are independent of each other.

In any case, we would want to execute the join, natural join, union, & intersection operators for two relations not more than once. In general, for queries that act on the same set of rows (might be different projections), these three ideas for building indices could be used:

1. Create an index on the Key or the unique value in those rows.
2. Create indices on the most common fields used for access, e.g: if the selection is based on HOURS, have an index on hours, the index providing the relation key which can be used for the row selection & projection (number of indices and depth depending on the space available)
3. Two way indices on key and commonly used values in the WHERE clause

    - If the rows are created as a result of a join, pre-compute the join, build index on the join intermediate.

[D]

A General Approach for finding Index Creation Heuristics:

Find out which particular field is being utilized the most for accessing rows. Let's say, if all queries are on 'Bdate', then it would be ideal to have the index built on that. Memory space is the biggest bottleneck. Suppose, we require indices on 10 different fields. Depending on how much memory we have, we'll need to compute the number of levels to have for each index since it would be ideal to have multi-level indices here. If we had unlimited space, we could have B+ trees on everything: fields in a table, a list of possible joins and crosses, etc. Since we don't have endless memory, we need to find out which fields (key or non-key or composite attribute) should be used for building indices, and how many levels each of them should have. This is a pure strategy design and optimization problem. We need to compute the most optimal set of fields for the purpose of indexing and a max_depth for each of the proposed indices.

A set of possible indices which would help the queries formulated in part B:

Index Description	**Query it helps**
Secondary index on attribute Pname on table PROJECT	Q1 - q1, Q3 - q8
Secondary index on attribute Dnum on table PROJECT	Q1 - q2, Q3 - q8
Primary index on attribute Pnumber on table PROJECT	Q2 - q1, q2 & Q1
Secondary index on attribute Plocation on table PROJECT	Q2 - q2
Primary index on attribute Dnumber on table DEPT_LOCATIONS	Q2 - q3, q4
Primary index on attribute Dlocation on table DEPT_LOCATIONS	Q2 - q3, q4
Primary index on attribute Dnumber on table DEPARTMENT	Q2 - q5
Primary index on attribute Ssn on table EMPLOYEE	Q3 - q1 thru q6
Secondary composite index on attribute (Fname, Lname) on table EMPLOYEE	Q3 - q3, q5 & q6
Secondary index on attribute Fname on table EMPLOYEE	Q3 - q1
Secondary index on attribute Lname on table EMPLOYEE	Q3 - q2
Secondary index on attribute Dno on table EMPLOYEE	Q3 - q5, q6
Secondary index on attribute Dname on table DEPARTMENT	Q3 - q7
Primary index on attribute Essn on table DEPENDENT	Q3 - q9
Secondary index on attribute Sex on table DEPENDENT	Q3 - q9, q10

Again, all of the above indices might not be implemented practically in a multi-level fashion. The strategy needs to provide a set from the above list such that most queries can be made efficient.

[E]

Best case: There exists a query Q which requires the extraction of exactly one disk block and outputs the set of rows R. All other queries except for Q in the batch produce output rows which are unique proper subsets of R. In that, we need to access only one relation and all the query outputs are populated with exactly one traversal of the relation. Also, the query Q does not require expensive joins or crosses.

For example:

1. SELECT Essn from WORKS_ON

2. SELECT Essn from WORKS_ON WHERE Hours > 4.0
3. SELECT Essn from WORKS_ON WHERE Hours > 9.0
4. SELECT Essn from WORKS_ON WHERE Hours > 12.0
5. SELECT Essn from WORKS_ON WHERE Hours > 30.0

Worst Case: Each query needs to access a separate relation altogether. In total, the queries cover all relations in the database, require large joins to estimate correct answers, queries can involve matching non-key fields (need to traverse the entire relation to make sure we are considering everything). And the queries involve different relations with nothing particular in common: i.e pre-computing or intermediate result saving does not really help.

For example:

1. SELECT * FROM EMPLOYEE INNER JOIN DEPARTMENT WHERE Dno=Dnumber
2. SELECT * FROM DEPENDENT
3. SELECT * FROM WORKS_ON
4. SELECT * FROM PROJECT
5. SELECT * FROM DEPT_LOCATIONS

[F]

In query processing and transaction management, having good optimization schemes and execution strategies are of utmost importance. Grouping queries in brackets that require only one relation access, query ordering for execution, and building indices on intermediate results are also important for processing batch queries. Understanding which queries require just one relation and populating those results with one access, running queries accessing independent relations in parallel are other important strategies.

In parts C and D, we saw how grouping queries based on which rows they access and building necessary indices for running queries more efficiently help mitigate some of the potential issues we saw in part A. Also, in part A, we went over strategies on a very broad level. Parts C and D allowed us to explore specific aspects in more detail.