Dr. N. SHAKEELA

Guest Lecturer

School of Computer Science, Engineering & Applications

Bharathidasan University

Trichy-24

Query Processing is the activity performed in extracting data from the database. In query processing, it takes various steps for fetching the data from the database. The steps involved are:

1. Parsing and translation
2. Optimization
3. Evaluation

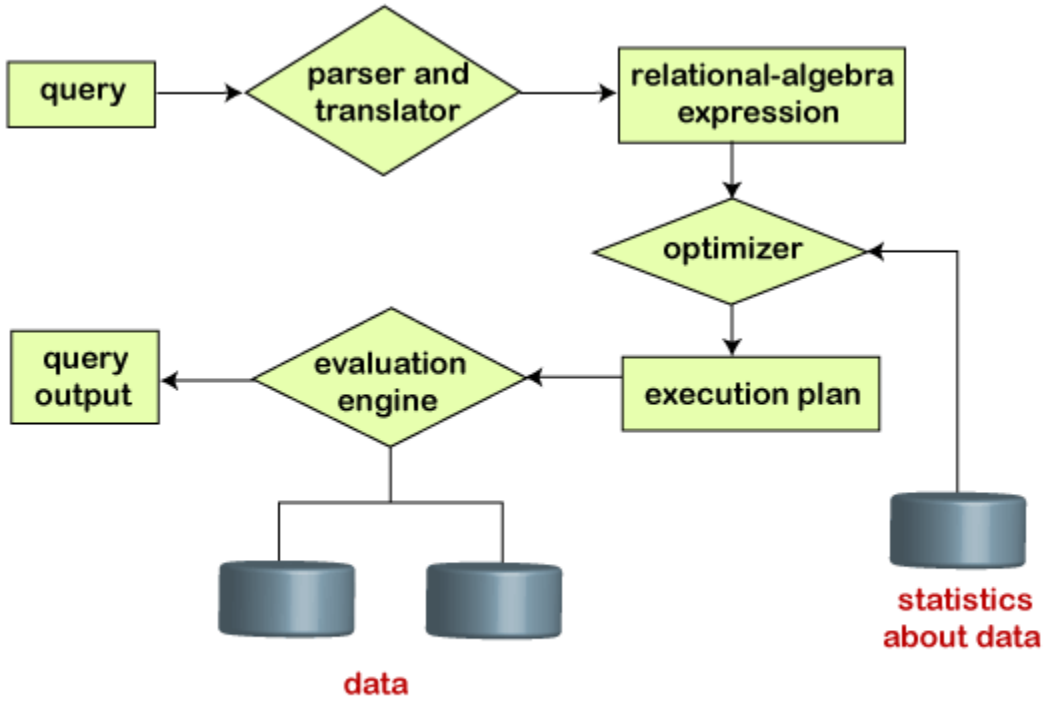
The query processing works in the following way:

Parsing and Translation

As query processing includes certain activities for data retrieval. Initially, the given user queries get translated in high-level database languages such as SQL. It gets translated into expressions that can be further used at the physical level of the file system. After this, the actual evaluation of the queries and a variety of query -optimizing transformations and takes place. Thus before processing a query, a computer system needs to translate the query into a human-readable and understandable language. Consequently, SQL or Structured Query Language is the best suitable choice for humans. But, it is not perfectly suitable for the internal representation of the query to the system. Relational algebra is well suited for the internal representation of a query. The translation process in query processing is similar to the parser of a query. When a user executes any query, for generating the internal form of the query, the parser in the system checks the syntax of the query, verifies the name of the relation in the database, the tuple, and finally the required attribute value. The parser creates a tree of the query, known as 'parse-tree.' Further, translate it into the form

of relational algebra. With this, it evenly replaces all the use of the views when used in the query.

Thus, we can understand the working of a query processing in the below-described diagram:



Steps in query processing

select emp_name from Employee where salary>10000;

Thus, to make the system understand the user query, it needs to be translated in the form of relational algebra. We can bring this query in the relational algebra form as:

- $\sigma_{salary>10000}$ (π_{salary} **(Employee))**
- π_{salary} ($\sigma_{salary>10000}$ **(Employee))**

After translating the given query, we can execute each relational algebra operation by using different algorithms. So, in this way, a query processing begins its working.

Evaluation

For this, with addition to the relational algebra translation, it is required to annotate the translated relational algebra expression with the instructions used for specifying and evaluating each operation. Thus, after translating the user query, the system executes a query evaluation plan.

Query Evaluation Plan

- In order to fully evaluate a query, the system needs to construct a query evaluation plan.
- The annotations in the evaluation plan may refer to the algorithms to be used for the particular index or the specific operations.
- Such relational algebra with annotations is referred to as **Evaluation Primitives**. The evaluation primitives carry the instructions needed for the evaluation of the operation.
- Thus, a query evaluation plan defines a sequence of primitive operations used for evaluating a query. The query evaluation plan is also referred to as **the query execution plan**.
- A **query execution engine** is responsible for generating the output of the given query. It takes the query execution plan, executes it, and finally makes the output for the user query.

Optimization

- The cost of the query evaluation can vary for different types of queries. Although the system is responsible for constructing the evaluation plan, the user does need not to write their query efficiently.
- Usually, a database system generates an efficient query evaluation plan, which minimizes its cost. This type of task performed by the database system and is known as Query Optimization.
- For optimizing a query, the query optimizer should have an estimated cost analysis of each operation. It is because the overall operation cost depends on the memory allocations to several operations, execution costs, and so on.

Finally, after selecting an evaluation plan, the system evaluates the query and produces the output of the query.

Measures of Query Cost

In DBMS, the cost involved in executing a query can be measured by considering the number of different resources that are listed below;

- The number of disk accesses / the number of disk block transfers / the size of the table
- Time taken by CPU for executing the query

The *time taken by CPU is negligible* in most systems when compared with the number of disk accesses.

If we consider the number of block transfers as the main component in calculating the cost of a query, it would include more sub-components. Those are;
Rotational latency – time taken to bring and spin the required data under the read-write head of the disk.
Seek time – time taken to position the read-write head over the required track or cylinder.
Sequential I/O – reading data that are stored in contiguous blocks of the disk
Random I/O – reading data that are stored in different blocks that are not contiguous.

That is, the blocks might be stored in different tracks, or different cylinders, etc.
Whether read/write? – read takes less time, write takes more.

From these sub-components, we would list the components of a more accurate measure as follows;

- The number of seek operations performed
- The number of block read
- The number of blocks written

To get the final result, these numbers to be multiplied by the average time required to complete the task. Hence, it can be written as follows;

Query cost = (number of seek operations X average seek time) +

(number of blocks read X average transfer time for reading a block) +

(number of blocks written X average transfer time for writing a block)

Note: here, CPU cost and few other costs like cost of writing the final result are omitted.

Selection Operation

Generally, the selection operation is performed by the file scan. **File scans** are the search algorithms that are used for locating and accessing the data. It is the lowest-level operator used in query processing.

Basic algorithms

- **A1: Linear Search:** In a linear search, the system scans each record to test whether satisfying the given selection condition. For accessing the first block of a file, it needs an initial seek. If the blocks in the file are not stored in contiguous order, then it needs some extra seeks. However, linear search is the slowest algorithm used for searching, but it is applicable in all types of cases. This algorithm does not care about the nature of selection, availability of indices, or the file sequence. But other algorithms are not applicable in all types of cases.
- **A2: Binary search:** IF the file is ordered on an attribute, and the selection condition is an equality comparison on the attribute, we can use a binary search to locate records that satisfy the selection.

Selection using indices:

The index-based search algorithms are known as **Index scans**. Such index structures are known as **access paths**. These paths allow locating and accessing the data in the file. There are following algorithms that use the index in query processing:

- **A3:Primary index, equality on a key:** We use the index to retrieve a single record that satisfies the equality condition for making the selection. The equality comparison is performed on the key attribute carrying a primary key.
- **A4:Primary index, equality on nonkey:** The difference between equality on key and nonkey is that in this, we can fetch multiple records. We can fetch multiple records through a primary key when the selection criteria specify the equality comparison on a nonkey.

- **A5:Secondary index, equality on key or nonkey:** The selection that specifies an equality condition can use the secondary index. Using secondary index strategy, we can either retrieve a single record when equality is on key or multiple records when the equality condition is on nonkey. When retrieving a single record, the time cost is equal to the primary index. In the case of multiple records, they may reside on different blocks. This results in one I/O operation per fetched record, and each I/O operation requires a seek and a block transfer.

Selection involving comparisons:

For making any selection on the basis of a comparison in a relation, we can proceed it either by using the linear search or binary search via indices in the following ways:

A6:Primary index, comparison: When the selection condition given by the user is a comparison, then we use a primary ordered index, such as the primary B^+-tree index. **For example**, when A attribute of a relation R compared with a given value v as A>v, then we use a primary index on A to directly retrieve the tuples. The file scan starts its search from the beginning till the end and outputs all those tuples that satisfy the given selection condition. For A>v, the file scan starts with the first tuple such that A>v.

For A<v we use simple file scan starting from the beginning of the file., and continuing up to the first tuple with attribute A=v.

- **A7:Secondary index, comparison:** The secondary ordered index is used for satisfying the selection operation that involves <, >, ≤, or ≥ In this, the files scan searches the blocks of the lowest-level index.
  (< ≤): In this case, it scans from the smallest value up to the given value v.
  (>, ≥): In this case, it scans from the given value v up to the maximum value.
  However, the use of the secondary index should be limited for selecting a few records. It is because such an index provides pointers to point each record, so users can easily fetch the record through the allocated pointers. Such retrieved records may require an I/O operation as records may be stored on different blocks of the file. So, if the number of fetched records is large, it becomes expensive with the secondary index.

Implementation of complex selections:

Working on more complex selection involves three selection predicates known as Conjunction, Disjunction, and Negation.

Conjunction: A conjunctive selection is the selection having the form as:

$$\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \wedge \theta n}(r)$$

A conjunction is the intersection of all records that satisfies the above selection condition.

Disjunction: A disjunctive selection is the selection having the form as:

$$\sigma_{\theta 1 \vee \theta 2 \vee \ldots \vee \theta n}(r)$$

A disjunction is the union of all records that satisfies the given selection condition θ_i.

Negation: The result of a selection $\sigma_{\neg\theta}(r)$ is the set of tuples of given relation r where the selection condition evaluates to false. But nulls are not present, and this set is only the set of tuples in relation r that are not in $\sigma_{\theta}(r)$.

Using these discussed selection predicates, we can implement the selection operations by using the following algorithms:

- **A8: Conjunctive selection using one index:** In such type of selection operation implementation, we initially determine if any access path is available for an attribute. If found one, then algorithms based on the index will work better. Further completion of the selection operation is done by testing that each selected records satisfy the remaining simple conditions. The cost of the selected algorithm provides the cost of this algorithm.
- **A9: Conjunctive selection using Composite index:** A composite index is the one that is provided on multiple attributes. Such an index may be present for some conjunctive selections. If the selection specifies an equality condition on two or more attributes, and a composite index exists on these combined attribute fields, then the index can be searched directly. Such type of index determine the suitable index algorithms.

- **A10: Conjunctive selection by the intersection of identifiers:** This implementation involves record pointers or record identifiers. It uses indices with the record pointers on those fields which are involved in the individual selection condition. It scans each index for pointers to tuples satisfying the individual condition. Therefore, the intersection of all the retrieved pointers is the set of pointers to the tuples that satisfies the conjunctive condition. The algorithm uses these pointers to fetch the actual records. However, in absence of indices on each individual condition, it tests the retrieved records for the other remaining conditions.

  A11 : Disjunctive selection by the union of identifiers: This algorithm scans those entire indexes for pointers to tuples that satisfy the individual condition. But only if access paths are available on all disjunctive selection conditions. Therefore, the union of all fetched records provides pointers sets to all those tuples which satisfy or prove the disjunctive condition. Further, it makes use of pointers for fetching the actual records. Somehow, if the access path is not present for anyone condition, we need to use a linear search to find those tuples that satisfy the condition. Thus, it is good to use a linear search for determining such tests