Database Management System

UNIT - 5

Query processing and Optimization

Outline….

- Evaluation of relational algebra expressions
- Query equivalence
- Join strategies
- Query optimization algorithms

Todays Topic…

- Domain and data dependency

Introduction to Query Processing

- Relational query processing refers to the range of activities which includes in extracting data from a database using a database query.
- A query processing involves below steps.
  1. Scan
  2. Parse
  3. Validate

1. Scan

- The scanner reads the language token such as SQL keywords, relation names in the text of the query.

2. Parse

- The parser check the query syntax to verify it is as per the syntax rules of the query language.

Introduction to Query Processing

3. Validate

- The query must also be validated by checking that all attributes and relation name are valid in the schema of the particular database being queried by query.
- **Query execute plan**
- Query executing plan will give idea about how query will be executing in stepwise manner.
- An internal representation of the query can be created as a tree data structure which is called a **query tree**.
- The DBMS must find all alternative execution strategies for retrieving the result of the query from the database.

Introduction to Query Processing

Example: Department information can be accessed in following ways:

1. Department table
2. Employee_Department_Join table

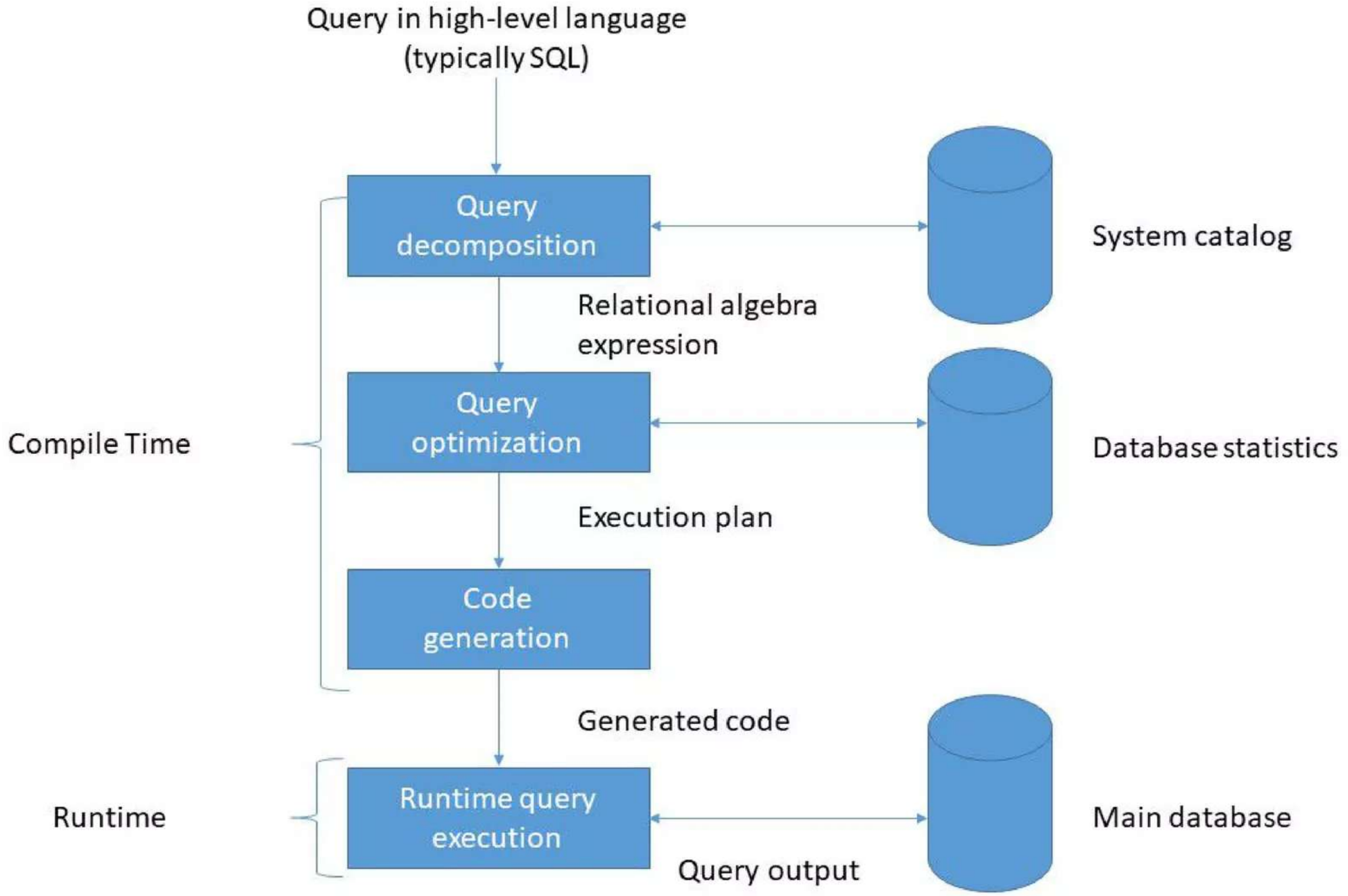
- A query can have many possible execution strategies.
- **Process of selection a suitable strategy of processing a query is known as query optimization.**
- Query optimization is achievable for simple queries but it becomes very complex for difficult queries.
- Each DBMS has a number of general database access algorithms that such as selection, projection or join or combination of these operations.

Typical Query Processing in DBMS

1. Introduction

- The scanning, parsing and validating module produces an internal representation of the query like query tree or query graph.
- The DBMS must then find its execution strategy for fetching results of query from the database files.
- A query typically can have many possible execution strategies the process of selecting a suitable one for processing a query is know as **query optimization**.

Typical Query Processing in DBMS



Typical Query Processing in DBMS

2. Step of query processing
   a) Query decomposition
   b) Query optimization
   c) Code generation
   d) Runtime query execution

a) Query decomposition

- In query decomposition include scanning, parsing and validating process.

I. Scanning

- The scanner find the language tokens like SQL keywords, attribute names, and relation name in the text of the query.

Typical Query Processing in DBMS

II. Parsing

- Parser checks the syntax of query to find out whether it is written according to the syntax rules (rules of grammar) of the query language.

iii. Validating

- Query validation is done by checking that all attribute (column) and relation (table) names are valid i.e. Table, column name present in given database.
- Semantic meaningful names in the given database schema which is being queried.
- An internal representation of the query is then shown as a tree data structure called a query tree.
- An internal representation of the query is then shown as a graph data structure called a query graph.

Typical Query Processing in DBMS

b) Query optimization

- Query optimization is one of the most important tasks of a relational DBMS.
- The query optimizer generates alternative plans and chooses the best plan with the latest estimated cost.
- The query optimizer module is used to find out an execution plan which is the execution strategy to retrieve the result of the query from the database files.
- A query can have many possible execution strategies each with different performances the process of selecting a reasonably efficient execution plan is known as query optimization.

Typical Query Processing in DBMS

c) Code generation

- The code generator generates the code to execute the execution plan selected by query optimization block.

d) Runtime query execution

- The above generated code will be accepted by runtime query processor which executes the generated code and produces the query result.
- This is desired query output that user wants.

Selection operation

- **Symbol**: σ (Sigma)
- **Notation**: $\sigma_{condition}(Relation)$
- **Operation**: Selects tuples from a relation that satisfy a given condition.

RollNo	Name	Branch	SPI
101	Raj	CE	8
102	Meet	ME	9
103	Harsh	EE	8
104	Punit	CE	9

$$\sigma_{Branch='CE'}(Student)$$

RollNo	Name	Branch	SPI
101	Raj	CE	8
104	Punit	CE	9

Search algorithm for selection operation

1. Linear search (A1)
2. Binary search (A2)

Linear search (A1)

- It **scans each blocks** and **tests all records** to see whether they **satisfy the selection condition**.
  - Cost of linear search (worst case) = **b_r**
    b_r denotes number of blocks containing records from relation r
- If the **selection condition is there on a (primary) key attribute**, then **system can stop searching if the required record is found**.
  - cost of linear search (best case) = **(b_r /2)**
- Linear search can be applied regardless of
  - **selection condition** or
  - **ordering of records in the file (relation)**
- This algorithm is **slower** than binary search algorithm.

Binary search (A2)

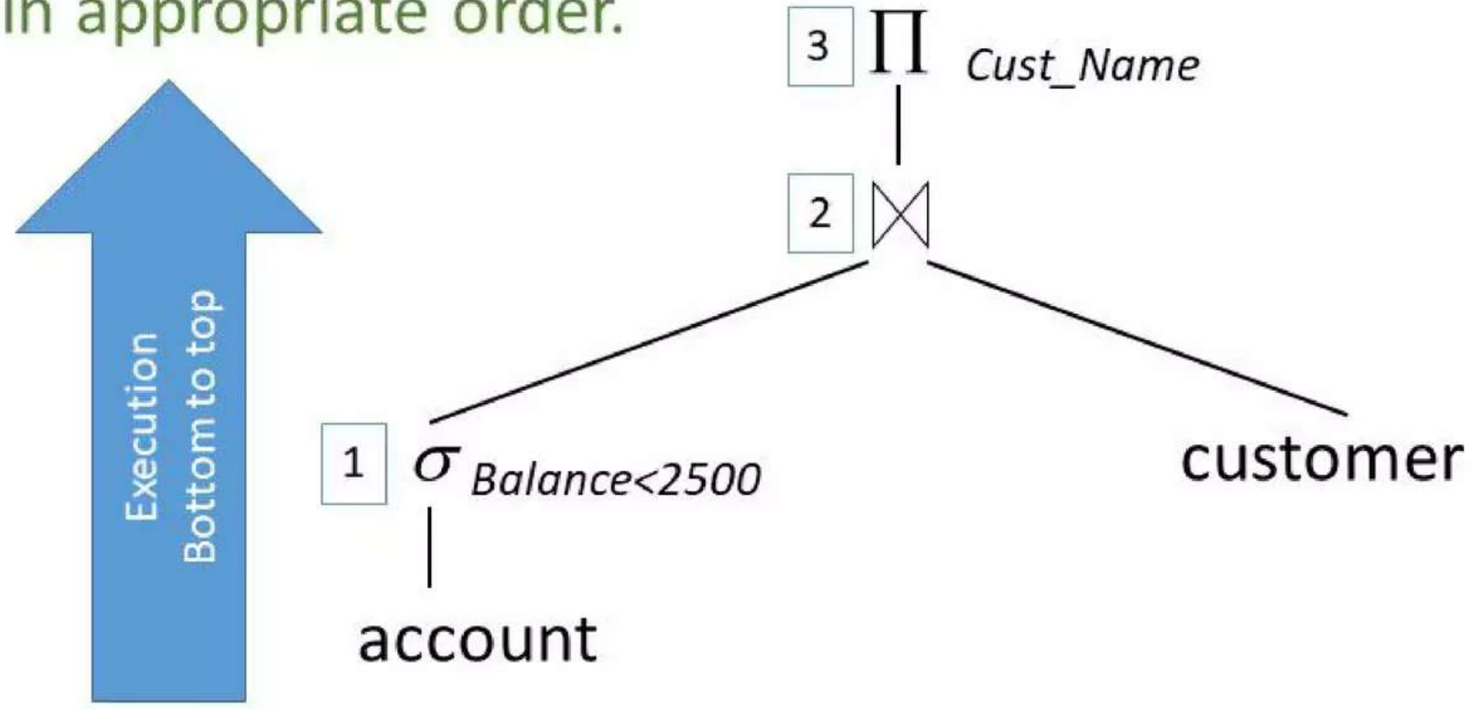
- Generally, this algorithm is used if **selection is an equality comparison on the (primary) key attribute** and **file (relation) is ordered (sorted) on (primary) key attribute**.
- cost of binary search = $\mathbf{[\log_2(b_r)]}$
  - b_r denotes number of blocks containing records from relation r
- If the selection is on **non (primary) key attribute** then **multiple block may contains required records**, then the **cost of scanning such blocks need to be added to the cost estimate**.
- This algorithm is **faster** than linear search algorithm.

Evaluation of expressions

- Expression may contain more than one operations, solving expression will be difficult if it contains more than one expression.

$$\Pi_{Cust\_Name}(\sigma_{Balance<2500}(account) \bowtie customer)$$

- To evaluate such expression we need to evaluate each operation one by one in appropriate order.



Algorithms for implementing Selection Operation with Indexes

- Introduction to indexing
- Index structures are referred to as access paths for set of data, since they provide a path through which data can be located and accessed.
- It is efficient to read the records of a file in an order closely to physical order.
- Primary index is an index that allows the records of a file to be read in the physical order in which they are strode in the file.
- All indices which are not a primary index is called a secondary index.
- Indices provide fast, direct and ordered access to data, But they impose the overhead of access to those blocks containing the index.

The search algorithm that uses equality operation with indexing

- 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:
- **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.
- **Primary index, equality on non key:** The difference between equality on key and non key 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 non key.

The search algorithm that uses equality operation with indexing

- **Secondary index, equality on key or non key:** 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 non key.
- 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.

The search algorithm that uses comparison operation with indexing

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

The search algorithm that uses comparison operation with indexing

- **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.

The Complex Search algorithms

1. **Conjunction:** A conjunctive complex selection is a selection of the form $\sigma \theta 1 \Lambda \theta 2 \Lambda \ldots \Lambda \theta n (R)$.
2. **Disjunction**: A disjunctive selection is a selection of the form $\sigma \theta 1 V \theta 2 V \ldots V \theta n (R)$.

- We can implement a selection operation involving either a conjunction or disjunction of simple conditions by using one of the following algorithms:

a. Conjunctive selection using one index.

b. Conjunctive selection using composite index.

c. Conjunctive selection by intersection of identifiers.

Conjunctive selection using one index

a. Conjunctive selection using one index (A_8)

- First we find whether an access path is available for an attribute in one of the simple conditions out of given set of conditions.
- After retrieving records satisfying simple condition then only algorithm looks for weather it matches next condition also or not.
- We complete the operation by testing in the memory buffer, whether or not each retrieved record satisfies the remaining simple condition.
- To reduce the cost, we choose θ_i and one of the algorithm A1 to A7 for which the combination results in the least cost of and $\sigma\theta$ * (R).

Conjunctive selection using one index

b. Conjunctive selection using composite index (A_9)

- Specific composite index may be available for some conjunctive selections.
- If the selections specifies some equality on two or more attributes and if composite index exists on these combined attribute fields, then the index can be searched directly.

c. Conjuctive selection by intersection of identifiers (A_{10})

- For this algorithm we require indices with record pointers on the fields which involved in the some individual conditions.
- The algorithm scans each and every index for all pointers to find tuples that satisfy condition. The intersection of the retrieved pointers is a the set of pointers to tuples that satisfy the conjunctive condition.

Conjunctive selection using one index

- The algorithm also uses the pointer to retrieve records.
- The cost of this algorithm is the sum of the costs of the individual index scans, and also the cost of retrieving the records in the intersection of the retrieved list of pointers.
- This cost can be reduced by sorting the list of pointers and retrieving records in sorted order.

Sorting

- Introduction
- Sorting of data is important in database management systems for two reasons.
- Some SQL queries specify that the output should be sorted.
- Second, and equally important for query processing.
- Relational operations like join can be implemented efficiently if the input relations are already sorted.
- For sorting a relation we can build an index on the sort key, and then using that index to read the relation in sorted order.
- However, such a process can order the relation only logically, through an index, rather than physically.

Types of Sorting

- There are two types of sorts :

1. Internal sorting

- Processing happens in memory.

2. External sorting

- Use this when you cannot fit the relation in memory so processing happens out of memory.
- Assume there are N memory buffers.

Types of Sorting

- Example: External merge sort
- There are two phase for external merger sort :

1. Phase 1 : Create sorted runs

- Fill the N memory buffers with the next N number of blocks of the relation.
- Sort these N blocks.
- Write the sorted blocks to disk.

2. Phase 2 : Merger sorted runs

- Assume there are at most N-1 runs.
- Read the first block of each run into memory.
- Each iteration finds the lowest record from the N-1 blocks.
- Place it into the memory buffer.
- If any block is empty, read its next block.

Join Operations / Join Strategies

- Concept of joins
- The relational operations can merges columns from multiple tables to form new (called joined table) table.

Table 1		
A	B	C

Table 2		
C	D	E

Joined Table				
A	B	C	D	E

- We can join multiple tables with help of some join condition (Equality or inequality) or without any join condition (cross join).
- Sorting of data is important operation in database systems for two reasons.
- Join is a time-consuming operation.

Nested Loop Join

- The nested loops join are also referred as nested iteration.
- The nested loops join accepts two inputs :
- Outer input table
- Inner input table
- The outer loop executed on the outer input table row by row and inner loop executed once for each outer row and scans row to search desired tuples.
- This search scans entire table or index so it is also called a naive nested loops join.
- A nested loops join is particularly effective if the outer input is small and the inner input is pre indexed and large.

Nested Loop Join – Working

I. Worst case

- The buffer can hold only one block of each relation.
- A total of nR * bs + br block accesses would be required.
- Where
- br = Number of blocks containing tuples of R.
- bs = Number of blocks containing tuples of S.

Nested Loop Join – Working

II. Best case

- There is enough space for both relations to fit simultaneously in memory so, each block would have to be read only one, hence only br + bs block accesses would be replaced.
- If one of the relations fits entirely in memory, it is beneficial to use that relation as the inner relation since the inner relation would then be read only once.

Blocks Nested Loop Join

- If the available buffer is very small to hold any of the relation entirely in memory still it is possible to obtain a major saving in block accesses, if we process the relations on a per block basis rather than per tuple basis.
- The block nested loop join algorithm improves performance of simple nested loop join by only scanning R once for every block of N tuples.
- This algorithm is a variable of nested loop join, where every block of the inner relation is paired with every block of the outer relation.
- Consider the natural join of Lecture and Student as discussed below.

Lecture |X| Student

Blocks Nested Loop Join – Working

I. Worst case

- Let us assume M pages of main memory. If M > 100, the join can be done in 400 + 100 disk accesses using plain nested – loop join.
- If R1 is outer relation,
- So we only consider the case where M < = 100 pages.
- Number of block access = (100 / 1) * 400 + 100

  = **40,100**

II. Best Case

- Number of block access = $B_{Student} + B_{Lecture}$

  = 100 + 400 = **500**

Indexed Nested Loop

- In a nested loop join if an index is available on the inners loop join attribute, index look ups can replace files scans in order to improve performance.
- For each tuple tR in the outer relation R, the index is used to look up tuples in table S that will satisfy the join condition with tuple tR.
- This join method is called indexed nested loop join, it can be used with existing indices as well as with temporary indices creating for the sole purpose of evaluating the join.

Indexed Nested Loop – Cost calculation

- In the worst case, there is space in buffer for only one page of relation R and one page of index.
- A total of bR + nR * c block access would be required.

Where,

bR = Number of blocks containing tuples of R.

nR = Number of tuples in R

c = cost of a single selection on s using the join condition

Merge Join

- Introduction
- The merge join algorithm is also called as sort merge join algorithm.
- The merge join algorithm is generally used to compute natural join and equi-joins.
- The merge join algorithm relates one pointer with each relation. These pointers point initially to the first tuple of the corresponding relations.
- As the algorithm processed, the pointers move through the relation.
- It is also possible to perform merge join operation on unsorted tuples, if secondary indices exist on both joining attributes.
- The algorithm scans the records through the indices and retrieved in sorted order.

Merge join – Cost calculation

- As merge join method makes a single pass through both files hence it is efficient,
- Number of block accesses = $b_r + b_s$

Where,

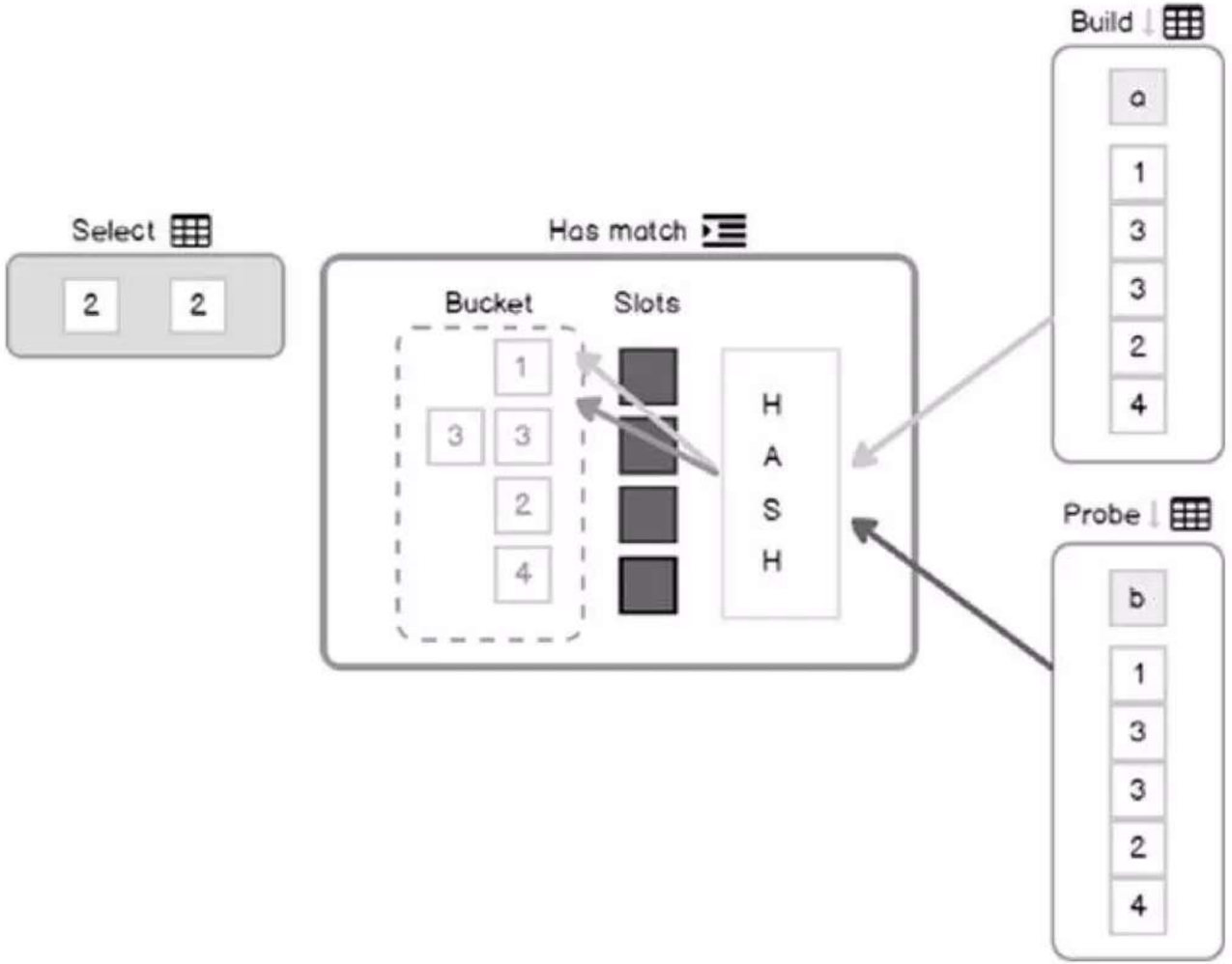
b_r = Number of blocks containing tuples of R.

b_s = Number of blocks containing tuples of S.

Hash Join

- Introduction
- The hash join is useful, if there are no adequate indexes on the joined columns. This is a worst situation. In this case, hash table will be created.
- This algorithm can also be used to implement natural joins and equi-joins. In this algorithm, a hash function H is used to partition tuples of both relations.
- The basic idea is to partition the tuples of each of the relations into sets that have same hash value on the join attributes.
- Hash join is most efficient when one of the tables is significantly different in size than other table.
- The query optimizer makes a hash join in two phases build phase, probe phase.

Hash Join - Working



Hash Join - Phases

- Build phase
- Hash table will be created by scanning each value in the build input and applying the hashing algorithm to the key.
- Joining attribute (column that join the tables) is called hash key.
- The hash table consists of lined list called hash buckets.
- The result of using a hash function on a hash key is called hash value.
- Hash value and RID (unique row identifier) will be placed into hash table.
- Hash value must be smaller than hash key. So query processor economies on the size of the hash table.

Hash Join - Phases

- Probe phase
- The entire probe input is scanned, and for each probe row computes the same hash value on the hash key to find any matches in the corresponding hash bucket.

Hash join – Cost calculation

- In the worst case, there is space in buffer for only one page of relation R and one page of index.
- The partitioning of the two relations R and S reads both relations completely and then write it back on them.
- Number of block accesses = $2 (b_r + b_s)$

Where,

b_r = Number of blocks containing tuples of R.

b_s = Number of blocks containing tuples of S.

Cost of Computing for all Joins

- Nested-loop join: $b_R + n_R * b_S$
- Block-nested loop join: $b_R + n_R * b_S$
- Index-nested loop join: $b_R + t_R * c$, where c is the cost of one index lookup.
- Sort-merge join: $b_R + n_R *$ sort_cost
- Hash-join: $3 * (b_R + n_R)$

Query Tree – Query Evalution Plan / Expression Evalution

- Introduction

a. A query tree is a tree structure that represents a relational algebra expression :

- Each leaf node represents an input relation.
- Each internal node represents a relation obtained by applying one relational operational operator to its child nodes.
- The root relation represents the query solution.

b. Two query trees are equivalent if their root relations i.e. Query solution are same.

c. A query tree may have multiple execution plans. Out of such query plans are more efficient than others.

Query Tree – Query Evalution Plan / Expression Evalution

d. For query tree we need SQL queries to be translated into extended relational algebra.

e. The process of finding a good evaluation plan is called query optimization.

Example

a. Consider Following SQL Query

```
SELECT S.Stud_Name FROM Student S, Lecturer L
WHERE S.Lid = L.Lid AND L.Name = 'abc' AND S.Branch='IT';
```

b. Lets find out relational algebra expression for the query :

- Find the names of all students who taught by lecturer abc and he is the IT branch.

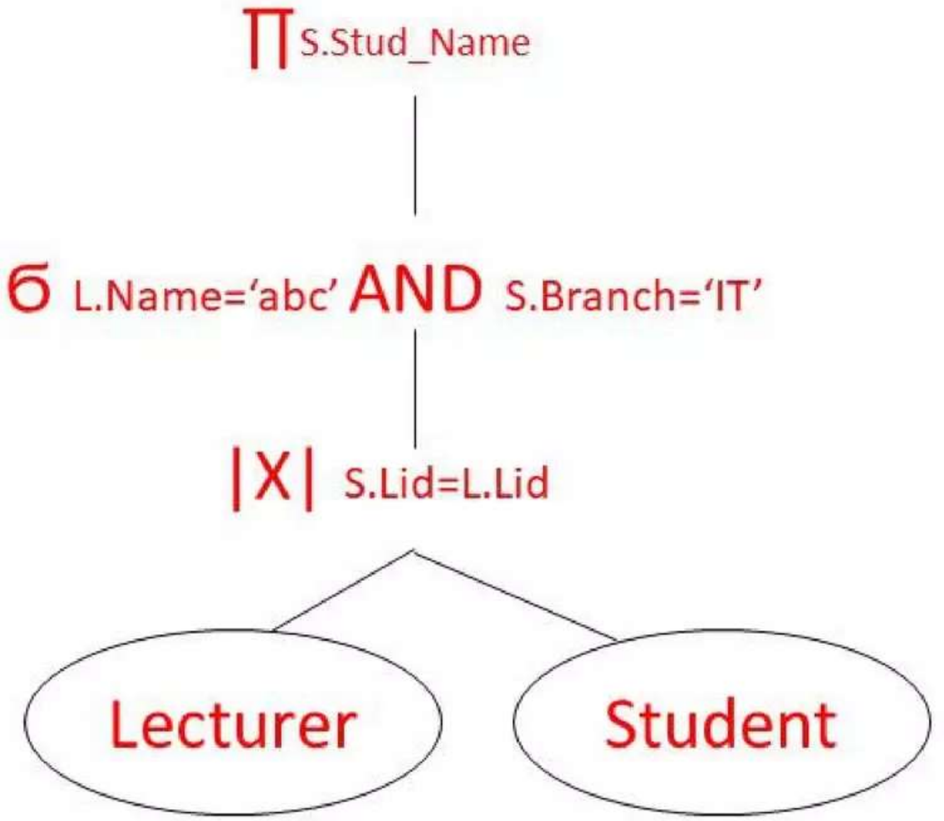
$$\prod_{S.Stud\_Name} (\sigma_{L.Name='abc' \text{ AND } S.Branch='IT'} (Lecture \ |X|_{S.Lid=L.Lid} \ Student))$$

- The above expression construct an intermediate relation as below,

$$Lecture \ |X|_{S.Lid=L.Lid} \ Student$$

Initial expression tree

- Even through we are interested in only those tuples pertaining to IT branch.

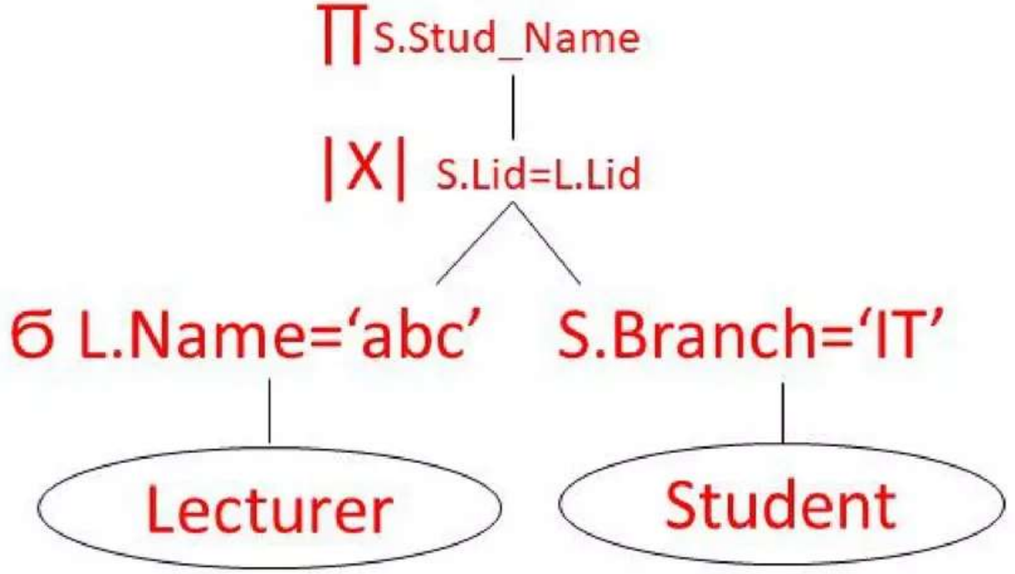


Transformed Expression

- We can reduce the number of tuples in student relation that we required to accessed, we can reduced the size of intermediate results.

$$\prod_{S.Stud\_Name}((\sigma_{L.Name='abc'}(Lecture\ L)) \bowtie_{S.Lid=L.Lid} (\sigma_{S.Branch='IT'}(Student\ S))$$

- Above expression is equivalent to our algebra expression, but it generates intermediate relations.



Query Tree – Query Evalution Plan / Expression Evalution

- To choose among different query evalution plans, the optimizer has to estimate the cost of each evalution plan. To compute the precise cost of evalution of a plan is usually not possible without actually evaluating the plan.
- Optimizer make use of system catalog and statistical information about the relations such as size of relation, index to estimate the cost of plan.

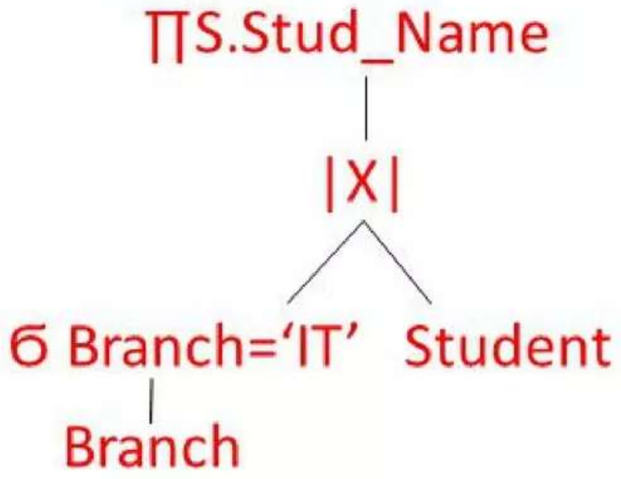
Evalution of relational algebra expression

- Introduction
- Expression may contain one or more operations, solving expression will difficult if it containing many expressions.
- After considering selection and join operations, now we need to consider how to evaluate an expression containing multiple operations.
- The simplest way to evaluate an expression is simply to evaluating only one operation at a time in an appropriate order. The result of each evaluation is stored in a temporary relation for next use.
- A disadvantage to this approach is the need to construct the temporary relations, which must be written to disk.
- The alternative approach is to evaluate several operations simultaneously in a pipeline with the result of one operation passed on to the next, in this case no need of a temporary relation.

Method 1: Materialization

- Introduction
- It is easiest to evaluate an expression by drawing a pictorial representation of the expression in form of an operator tree.
- Example
- Consider the expression

$$\prod S.Stud\_Name\ ((\sigma\ Branch='IT'\ (Branch))\ |X|\ Student)$$



Method 1: Materialization

- Operation
- In this approach we start from bottom of the tree.
- In above example, there is only one such operation, selection operation on branch table.
- The input to the lowest level operations are relations in the database.
- We execute these operations by the algorithms, which we studied in previous step and we store the results in temporary relations.
- Such temporary relations used to execute the operations at the next level up in the tree, where the inputs now are either temporary relations or relations stored the database.
- In example the inputs to join are the Student relation and the temporary relation created by the selection on branch.

Method 1: Materialization

- Operation
- The join can now be evaluated, creating another temporary relation.
- By repeating this process we can evaluate the operation at the root of the tree, giving the final result of the expression.
- Such evalution is called materialized evalution, as the result of each intermediate operation are created and then it is used for evaluation of the next level operation.

Method 1: Materialization

- Cost of evalution
- To compute the cost of evaluating an expression by adding the costs of all operations as well as the cost of writing intermediate results to disk.
- Cost of writing out the result = n_r / f_r

Where

- n_r = The estimated number of tuples in the result relation R
- f_r = The number of records of R that will fit in a block.

Method 2: Pipelining

- Operation
- To reduce number of intermediate temporary files produced by combining several relational operation into a pipeline of operation by passing result of one operation to the next operation in the pipeline.
- Combine operations into a pipeline eliminates the cost reading and writing temporary relations.
- Example
- Consider the expression :

$$(\prod (R \mid X \mid s)$$

- If we use materialization operation evalution would involve cresting a temporary relation to hold the result of the join and the reading back in the result to perform the projection.

Method 2: Pipelining

- Example
- Instead of above operations can be combined as whether join operation generates a result tuples it will be passes immediately to the project operation for processing.
- By combining the join and the projection we avoid creating the intermediate result and instead create the final result directly.
- Types

a) Demand driven pipeline

b) Producer driven pipeline

Method 2: Pipelining

a) Demand driven pipeline

- In this approach the system makes repeated requests for tuples from the operation at the top of the pipeline.
- Each time that an operation receiver a request for tuples it computes the next tuples to be returned and them returns that tuples.
- If the input operations are not pipelined the next tuple to be returned can be computed from the input relations, while the system keeps track of what has been returned so far.

Method 2: Pipelining

b) Producer driven pipeline

- The operations do not wait for request to produce tuples, but they it generate tuple.
- Each operation at the bottom of a pipeline continually generates output tuples and puts them in its out buffer, until buffer is full.
- Once the operation uses a tuple from a pipelined input it removes the tuple from its input buffer.

Transformation of relational expressions

- An SQL query is first translated into an equivalent relational algebra expression then it is represented as a query tree data structure which can be optimized later on.
- Step 1 : Decompose query into simple blocks
- SQL queries can be decomposed into small sized query blocks, which can be solved individually and which can be optimized further.
- Let us look at following example to understand this translation,
- Example
- Consider the following SQL query on the STUDENT relation

```
SELECT LNAME,FNAME FROM STUDENT
WHERE Fees < (SELECT MIN(Fees) FROM STUDENT WHERE DName='IT')
```

Transformation of relational expressions

- We can decomposed query into two blocks.
- Inner block
- To find minimum fees among all IT student

  SELECT MIN(Fees) FROM STUDENT WHERE DName='IT'

- Outer block
- To find student details whose fees greater then minimum fees among all students.

  SELECT LNAME,FNAME FROM STUDENT WHERE Fees < x

- Where x represent the result returned from the inner block

Transformation of relational expressions

- The query optimizer would then choose a best execution plan for each block. Inner block in above example needs to be evaluated only once to produce the minimum fees among IT students then used as the constant by the outer block such correlated queries are very complex for their execution.
- Step 2 : Convert query block to relational algebra equation
- The GROUP BY and HAVING are operation in the extended algebra used for plans, and that aggregate operations.
- T => ∏ min(fees) (б DName = 'IT' (Student))
- Final query => ∏ LNAME,FNAME (б Fees < T (Student))

Transformation of relational expressions

- Step 3 : Convert relational algebra expression into initial query tree
- The GROUP BY and HAVING are operation in the extended algebra used for plans, and that aggregate operations.