Access Path Selection in a Relational Database Management System

Jonathan Herman

Introduction

- System R published in 1976, 3 years prior
- System R uses SQL to perform queries
- SQL provides only a high level abstraction
- What order of operations leads to the most effective execution?

Query Block

- SELECT <items to be retrieved>
  FROM <tables items are located in>
  WHERE <conditions for item retrieval>
- May have nested query blocks and ordering

SQL Statement Processing

- Four phases
  - Parsing
  - Optimization
  - Code Generation
  - Execution

Optimization

- Performs semantic analysis on the SQL query
- Performs access path selection
  - Determines the fastest method of executing the query

Data Storage in System R
Types of Scans in RSS

- **Segment scan**
  - Tuples are stored on pages
  - Relations can span multiple pages
  - Can have multiple relations per segment

- **Index scan**
  - Use B-trees stored on separate pages than the actual tuples

Comparison of Scans

- **Segment scan** reads each page only once and must scan the entire relation
- **Index scan** may read from pages more than once, but lets you jump to a target tuple.
- Proximity of the data is not the same as the proximity of the indexes. Unless it's clustered

Optimizer Cost Formula

- Cost = Page Fetches + W * (RSI Calls)
- Page Fetches are used as a measure of I/O utilization
- W is an adjustable weight between I/O and CPU utilization
- The number of RSI calls is used to approximate CPU utilization

Selectivity

- The optimizer assigns a selectivity factor F for each boolean comparison
- The factor roughly corresponds with the percentage of tuples that satisfy the comparison
- Used to approximate the number of RSI calls when calculating the cost

Example – Bank Account

Query for all bank transactions
The transactions are in a single relation, stored on 2 pages and has a cardinality of 20 tuples
Assume a W value of 1 for simplicity
No selectivity, so F = the cardinality

Segment Scan: Cost = 2 + 1 * 20 = 2 + 20 = 22
Index Scan: Cost = 20 + 1 * 20 = 20 + 20 = 40

Example – Address

Query for an employee's home address
Again, assume the a single relation on 2 pages with a cardinality of 20 tuples
Assume a W value of 1 for simplicity
Use the F value in column = value

Segment Scan: Cost = 2 + 1 * (20 * 1/0) = 2 + 0 = 2
Index Scan: Cost = 1 + 1 * (20 * 1/20) = 1 + 1 = 2

Selectivity

- The optimizer assigns a selectivity factor F for each boolean comparison
- The factor roughly corresponds with the percentage of tuples that satisfy the comparison
- Used to approximate the number of RSI calls when calculating the cost

Example – Bank Account

Query for all bank transactions
The transactions are in a single relation, stored on 2 pages and has a cardinality of 20 tuples
Assume a W value of 1 for simplicity
No selectivity, so F = the cardinality

Segment Scan: Cost = 2 + 1 * 20 = 2 + 20 = 22
Index Scan: Cost = 20 + 1 * 20 = 20 + 20 = 40

Example – Address

Query for an employee's home address
Again, assume the a single relation on 2 pages with a cardinality of 20 tuples
Assume a W value of 1 for simplicity
Use the F value in column = value

Segment Scan: Cost = 2 + 1 * (20 * 1/0) = 2 + 0 = 2
Index Scan: Cost = 1 + 1 * (20 * 1/20) = 1 + 1 = 2
Single Relation Access Path

- For a single relation with no ordering, calculate the cost for each index and segment scan
- Choose the cheapest available option as the plan for the query block

Tuple Ordering

- Ordering clauses in the query block are labeled “interesting” by the optimizer
- The optimizer must decide when the query result must be sorted when it calculates the cost of the access paths
- Sorting can be done while performing the scans or after the scans as long as the correct ordering is produced

2-Way Joins

- Nested loops
  - A scan is chosen for the inner and outer relations of the join
  - The scans are performed, with tuple results given one at a time
  - For each tuple T in scan(outer)
    - For each tuple U in scan(inner)
    - Add Composite(T, U) to final result table

2-Way Joins (continued)

- Merging scans
  - Combines two relations on a single column
  - If there are multiple columns in the predicate list, choose one as the column to merge on
  - Becomes more efficient if the relations are clustered on the merging column (e.g. sorted on the column)

N-Way Joins

- A N-Way join is performed through several 2-way joins on the relations
- The order in which the joins are performed will determine how efficiently the joins are performed
- Relations should be ordered in cardinality increases to achieve the optimal ordering

N-Way Join Ordering

- A tree of possible join orderings is generated by the optimizer
- The tree must also consider the points at which the relations can be sorted by the “interesting” constraints
N-Way Join Cost

- The cost for performing a join is the same regardless of which method chosen
- N = The product of cardinalities of all relations T of the join so far * the product of the selectivity factors of all applicable predicates

N-Way Join Cost (continued)

- Cost(path1, path2) = Cost(path1) + N * Cost(path2)
- This takes into account the order of the joins through the N factor
- The cost of sorting the data may also be added to the path if the sorting is performed after the join

Optimizer Algorithm

- Calculate the Costs for using each index and a segment scan on each relation
- Generate a tree of the least cost access paths using the cost information
- Add join and sort information to the tree and choose the fastest traversal from root to leaf

Figure 1. JOIN example

Figure 3. Search tree for single relations
Nested Queries

- The WHERE clause of the query block contains another query block, called the inner query.
- The inner query must be executed before the outer query.
- If the inner query references a field in the outer query, it is executed each time the outer query is executed.

References


References (continued)