Access Path Selection in a Relational Database Management System

CMPT 843

Pouya Fattahi
Outlines

- How System R processes an SQL query?
Outlines

- How System R processes an SQL query?
- How System R optimize an SQL query?
Four steps to process an SQL query

◦ Parsing
◦ Optimization
◦ Code generation
◦ Execution
Parsing

- Each statement consists of one or more Query blocks

- Each statement has a predicate that may have one operand that is a query
Optimizer

Query express **what** data to retrieve,
**Not how** to retrieve it
Optimizer

Query express **what** data to retrieve,

*Not how* to retrieve it

The Database must pick the **best** execution strategy through a process known as optimization.
Optimizer

- **Example:**
  
  ```sql
  SELECT name, title, sal
  FROM Emp, Job
  WHERE Emp.Job = Job.Job
  and Title = 'CLERK'
  ```

- **Decide order to perform the different operators:**
  - process “Title = ‘CLERK’” followed by the join
  - Process the join “Emp.Job = Job.Job” followed by “Title = ‘CLERK’”
Code Generator

- Code generator translates ASL (output of optimizer) tree to executable machine code.

ASL

1. Store in DB for later
2. Execute

After executing

- Code calls upon the RSS (Research Storage System)
- Via RSI (Storage System Interface)
RSS (Research Storage System)

- Maintains physical storage of relations, access paths, locking, logging
- Relations are stored as a collection of tuples
- Tuples are stored on 4K pages; pages are organized into segments.
- Pages are organized into logical units called segments.
- Segments may contain one or more relations
  - Each tuple is tagged with the identification of the relation to which it belongs
- At most one relation per segment.
- Tuples are accessed via a scan: segment scan or index scan
Segment scan

- A series of NEXTs on a segment scan examines all pages of the segment which contain tuple.
- Returns those tuples belonging to the given relations
- All the non-empty pages of a segment will be touched
Index scan

- These indexes are stored on separate pages from those containing the relation tuples.
- Indexes are implemented as B-tree
HOW ?

- Enumerating the different execution plans,
- Estimate the cost of performing each plan,
- Pick the cheapest plan.

- What is definition of cost?
HOW?

- Enumerating the different execution plans,
- Estimate the cost of performing each plan,
- Pick the cheapest plan.

- What is definition of cost?
  - \( \text{COST} = \text{Page fetches} + W \times (\text{RSI Calls}) \)
Cost computation

- Formulates a cost prediction for each access plan, using the following cost formula:
  
  - \(\text{COST} = \text{Page fetches} + W \times (\text{RSI Calls})\)
  - \(\text{cost} = \text{IO costs} + W \times \text{CPU costs}\)

- \(W\) is an adjustable weighting factor between I/O and CPU.
- RSI calls is an approximation for CPU utilization.
Database Catalog
Statistics on the relations

- **For each relation** $T$
  - $NCARD(T)$, the cardinality of relation $T$
  - $TCARD(T)$, the number of pages in the Segment that hold tuples of relation $T$
  - $P(T)$, the fraction of data pages in the segment that hold tuples of relation $T$
    - $P(T) = \frac{TCARD(T)}{\text{# of non-empty pages in the segment}}$

- **For each index** $I$ on relation $T$
  - $ICARD(I)$, number of distinct keys in index
  - $NINDEX(I)$, the number of pages in index $I$

<table>
<thead>
<tr>
<th>StudentID</th>
<th>Lastname</th>
<th>Firstname</th>
<th>Gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Smith</td>
<td>John</td>
<td>M</td>
</tr>
<tr>
<td>102</td>
<td>Jones</td>
<td>James</td>
<td>M</td>
</tr>
<tr>
<td>103</td>
<td>Mayo</td>
<td>Ann</td>
<td>F</td>
</tr>
<tr>
<td>104</td>
<td>Jones</td>
<td>George</td>
<td>M</td>
</tr>
<tr>
<td>105</td>
<td>Smith</td>
<td>Suse</td>
<td>F</td>
</tr>
</tbody>
</table>

$NCARD(\text{StudentID}) = 5$
$NCARD(\text{Lastname }) = 3$
Next step in query Optimization

- Calculate a selectivity factor ‘F’ for each boolean factor in the predicate list
- For each relation, calculate the **cost** of scanning
- Find the efficient path for executing the query
Selectivity Factor

<table>
<thead>
<tr>
<th>Columns</th>
<th>Selectivity Factor</th>
</tr>
</thead>
</table>
| attr = value            | F = 1/ICARD(attr index) – if index exists
|                         | F = 1/10 otherwise                                     |
| attr1 = attr2           | F = 1/max(ICARD(I1),ICARD(I2)) or
|                         | F = 1/ICARD(ii) – if only index i exists, or
|                         | F = 1/10                                               |
| val1 < attr < val2      | F = (value2-value1)/(high key-low key)                  |
|                         | F = 1/4 otherwise                                      |
| expr1 or expr2          | F = F(expr1)+F(expr2)−F(expr1)*F(expr2)                |
| expr1 and expr2 F       | F = F(expr1) * F(expr2)                                |
| NOT expr                | F = 1 − F(expr)                                        |
Relation Cost

- Based on different situation like having index, clustered index, non-clustered index, group by or order by in our query block, the cost of relation will be calculated.
## Cost Formulas

<table>
<thead>
<tr>
<th>SITUATION</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique index matching an equal predicate</td>
<td>1 + 1 + W</td>
</tr>
<tr>
<td>Clustered index I matching one or more boolean factors</td>
<td>F(preds)* (NINDX(I)+TCARD) + W*RSICARD</td>
</tr>
<tr>
<td>Non clustered index I matching one or more boolean factors</td>
<td>F(preds)* (NINDX(I)+NCARD) + W*RSICARD</td>
</tr>
<tr>
<td>Clustered index I not matching any boolean factors</td>
<td>(NINDX(I) + TCARD) + W*RSICARD</td>
</tr>
</tbody>
</table>
Different number of relations

- Single relation
- 2-way join
- N-way join
Access path selection for joins

- **Outer relation**: Relation from which a tuple will be retrieved first

- **Inner relation**: Relation from which tuples will be retrieved, depending on the values obtained in the outer relation tuple.

- **Join predicate**: A predicate which relates columns of two tables to be joined.

- **Join column**: The column referenced in a join predicate
Join Method

1. **Nested Loops**: The scan on the outer relation is opened and the first tuple is retrieved. For each outer tuple obtained, a scan is opened on the inner relation to retrieve one at a time all tuples of inner relation that satisfy the join Predicate.

<table>
<thead>
<tr>
<th>Table</th>
<th>JOIN TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>range</td>
</tr>
<tr>
<td>T2</td>
<td>ref</td>
</tr>
<tr>
<td>T3</td>
<td>all</td>
</tr>
</tbody>
</table>

```
For each row in t1 matching range {
    For each row in t2 matching reference key {
        For each row in t3 {
            if row satisfies join conditions, do
        }
    }
}
```
Join Method

2. **Merging Scan**: Relations are scanned in join column order. Merging Scans method is only applied to equi-joins.
Cost of different Join

- The cardinality of the join of n relations is the same
- The cost of joining in different orders can be different
Order of join

If there are \( n \) relations then there are \( n! \) ways of joining.

In joining relations \( t_1, t_2, t_3, ..., t_n \) only those orderings \( t_{i1}, t_{i2}, t_{i3}, ..., t_{in} \) are examined in which for all \( j \) (\( j=2, ..., n \)) either

1. \( t_{ij} \) has at least one join predicate with some relation \( t_{ik} \), where \( k<j \) or
2. for all \( k > j \), \( t_{ik} \) has no join predicate with \( t_{i1}, t_{i2}, ..., \) or \( t_{i(j-1)} \)

**Example:** Let \( T_1, T_2, T_3 \) be relations such that there are join predicates between \( T_1 \) and \( T_2 \) and between \( T_2 \) and \( T_3 \) on different columns than on \( T_1-T_2 \) join then

\[
\begin{align*}
T_1 & \quad T_2 & \quad T_3 & \quad T_2 & \quad T_1 & \quad T3 & \quad T1 & \quad T2 \quad (\text{exclude}) \\
T1 & \quad T3 & \quad T2 \quad (\text{exclude}) & \quad T2 & \quad T3 & \quad T1 & \quad T3 & \quad T2 & \quad T1
\end{align*}
\]
Order of join

If there are $n$ relations then there are $n!$ ways of joining.
Order of join

If there are $n$ relations then there are $n!$ ways of joining.

A heuristic is used to reduce he join order permutations.
Computation of costs

- $C_{\text{outer}}(\text{path1})$ be the cost of scanning the outer relation via path 1, $N$ be the cardinality of the outer relation tuples which satisfy the applicable predicates:
  \[
  N = (\text{Product of cardinalities of all relations T of the join so far}) \times \\
  \text{(Product of selectivity factors of all applicable predicates)}
  \]

- $C_{\text{inner}}(\text{path2})$ be the cost of scanning the inner relation

- $C_{\text{nested loop join}}(\text{path1},\text{path2}) = C_{\text{outer}}(\text{path1}) + N \times C_{\text{inner}}(\text{path2})$

- $C_{\text{merge}}(\text{path1},\text{path2}) = C_{\text{outer}}(\text{path1}) + N \times C_{\text{inner}}(\text{path2})$

$C_{\text{sort}}()$ includes the cost of retrieving the data, sorting the data, which may involve several passes and putting the results into a temporary list.
### Example of Tree

<table>
<thead>
<tr>
<th>EMP</th>
<th>NAME</th>
<th>DNO</th>
<th>JOB</th>
<th>SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>50</td>
<td>12</td>
<td>8500</td>
<td></td>
</tr>
<tr>
<td>JONES</td>
<td>50</td>
<td>5</td>
<td>15000</td>
<td></td>
</tr>
<tr>
<td>DOE</td>
<td>51</td>
<td>5</td>
<td>9500</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPT</th>
<th>DNO</th>
<th>DNAME</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>50</td>
<td>MFG</td>
<td>DENVER</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>BILLING</td>
<td>BOULDER</td>
</tr>
<tr>
<td></td>
<td>51</td>
<td>SHIPPING</td>
<td>DENVER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JOB</th>
<th>TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>CLERK</td>
</tr>
<tr>
<td>6</td>
<td>TYPIST</td>
</tr>
<tr>
<td>8</td>
<td>SALES</td>
</tr>
<tr>
<td>12</td>
<td>MECHANIC</td>
</tr>
</tbody>
</table>

```sql
SELECT NAME, TITLE, SAL, DNAME
FROM EMP, DEPT, JOB
WHERE TITLE = "CLERK"
AND LOC = "DENVER"
AND EMP.DNO = DEPT.DNO
AND EMP.JOB = JOB.JOB
```
Example of Tree

EMP:  
| index  
| EMP.DNO 
| N₁  
| C(EMP.DNO)  

| EMP.JOB  
| N₁  
| C(EMP.JOB)  

| segment  
| scan on  
| EMP  
| N₁  
| C(EMP seg. scan)  

DEPT:  
| index  
| DEPT.DNO 
| N₂  
| C(DEPT.DNO)  

| Segment  
| Scan on  
| DEPT  
| N₂  
| C(DEPT seg. scan)  

| X  
| pruned  

JOB:  
| index  
| JOB.JOB 
| N₃  
| C(JOB.JOB)  

| segment  
| scan on  
| JOB  
| N₃  
| C(JOB seg. scan)  

EMP  
| NAME | DNO | JOB | SAL  
| SMITH | 50 | 12 | 8500  
| JONES | 50 | 5 | 15000  
| DOE | 51 | 5 | 9500  

DEPT  
| DNO | DNAME | LOC  
| 50 | MFG | DENVER  
| 51 | BILLING | BOULDER  
| 52 | SHIPPING | DENVER  

JOB  
| JOB | TITLE  
| 5 | CLERK  
| 6 | TYPIST  
| 9 | SALES  
| 12 | MECHANIC
Example of Tree

Figure 3. Search tree for single relations
Example of Tree

Figure 4. Extended search tree for second relation (nested loop join)
Example of Tree

Figure 5. Extended search tree for second relation (merge join)
Nested Queries

- The most deeply nested Subquery is evaluated before the main query and is evaluated only once.

```
SELECT NAME
FROM EMPLOYEE
WHERE SALARY =
    ( SELECT AVG(SALARY)
      FROM EMPLOYEE )
```

Return a single value

```
SELECT NAME
FROM EMPLOYEE
WHERE DEPNO IN
    ( SELECT DEPNO
      FROM DEPARTMENT
      WHERE LOCATION="DENVER" )
```

Return a set of value
Correlation Subquery

- Subquery may contain a reference to a value obtained from a tuple from higher level.

```sql
SELECT NAME
FROM EMPLOYEE X
WHERE SALARY > ( SELECT SALARY
    FROM EMPLOYEE
    WHERE EMPLOYEE_NUMBER=X.MANAGER)
```
Conclusion

- Database management systems can support non-procedural query languages with performance comparable to those supporting the current more procedural languages.
Q & A