CS 764: Topics in Database Management Systems
Lecture 6: Query Optimization

Xiangyao Yu
9/26/2022
Announcement

Updated schedule

10/12 Guest lecture from TiDB
... 
10/24 Guest lecture from Oracle
... 
11/7 Midterm review
11/9 Midterm exam

Course project

Proposal deadline is 10/24. Please start to form teams (2–4 people) asap.
Access Path Selection in a Relational Database Management System

R. Reissner, R. F. Goodman, C. A. Leslie, V. E. Price
IBM Research Division, San Jose, California 95193

1. Introduction

Our Project 8 is an experimental database management system being developed by IBM's San Jose Research Laboratory and supported in part by the U.S. Government. As of the time of the first release of the system in 1976, the software was developed by IBM research scientists, and the system is available to the IBM Research Division.

This paper concerns the selection of access paths for executing SQL statements in the experimental database management system described above. The system is based on the relational model of data and supports both procedural and nonprocedural ways of querying the database.

In the system, the user need not know how the tables are physically stored and what access paths are available in a given file. In fact, the system does not require the user to specify anything about the access path to be used for a given query. The system automatically selects the most efficient access path for each query and returns the result set.

2. Processing of an SQL statement

A SQL statement is submitted to the system by the user. The system, upon receiving an SQL statement, parses it and constructs a parse tree. The parse tree is then used to generate a query plan. The query plan is a tree structure that represents the order in which the subqueries are executed to produce the final result set.

The query plan is then used to select the most efficient access path for each subquery. Access paths are selected based on factors such as the size of the data set, the type of access path, and the expected cost of each access path.

Once the access paths are selected, the system executes the query plan to retrieve the desired data. The query plan is executed in a bottom-up manner, starting with the subqueries at the bottom of the parse tree and working up to the root query.

The system returns the result set to the user in the form of an SQL result set. The result set contains the data that satisfies the query criteria, organized in a tabular format. The user can then use the result set to further process the data, such as performing calculations or generating new tables.
Agenda

System R

Query Optimization in R
  • Cost estimation
  • Plan enumeration
System R

1. Parsing
2. Optimization
3. Code generation
4. Execution
Query Optimization

Query Parser

- SQL query
- unoptimized plan

Query Optimizer

- Plan generator
- Plan cost estimator

System Catalog

- optimized plan
Query Optimization in System R
System R Storage Architecture

Cost = IO cost + Computation cost
= #I/Os + W * RSICARD
RSICARD = #tuples through the RSI interface

Goal: enumerate execution plans and pick the one with the lowest cost
<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCARD(T)</td>
<td># tuples in T</td>
</tr>
<tr>
<td>TCARD(T)</td>
<td># of pages containing tuples in T</td>
</tr>
<tr>
<td>P(T)</td>
<td>Fraction of segment pages that hold tuples of T.</td>
</tr>
<tr>
<td>P(T) = TCARD(T) / # non-empty pages in the segment</td>
<td></td>
</tr>
<tr>
<td>ICARD(I)</td>
<td># distinct keys in the index I</td>
</tr>
<tr>
<td>NINDEX(I)</td>
<td># pages in index I</td>
</tr>
<tr>
<td>High key value and low key value</td>
<td></td>
</tr>
<tr>
<td>Modern systems</td>
<td>Keep histogram on table attributes.</td>
</tr>
</tbody>
</table>
Access Paths

Segment Scans
- A segment contains disk pages that can hold tuples from multiple relations
- Segment scan is a sequential scan of all the pages

<table>
<thead>
<tr>
<th>Page 1</th>
<th>A {...}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B {...}</td>
</tr>
<tr>
<td>Page 2</td>
<td>A {...}</td>
</tr>
<tr>
<td>Page 3</td>
<td>B {...}</td>
</tr>
<tr>
<td>Page 4</td>
<td>A {...}</td>
</tr>
<tr>
<td></td>
<td>B {...}</td>
</tr>
</tbody>
</table>
Access Paths

Segment Scans
- A segment contains disk pages that can hold tuples from multiple relations
- Segment scan is a sequential scan of all the pages

Index Scan
- Clustered index scan
- Non-clustered scan
- Scan with starting and stopping key values

<table>
<thead>
<tr>
<th>Page 1</th>
<th>A{...}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B{...}</td>
</tr>
<tr>
<td>Page 2</td>
<td>A{...}</td>
</tr>
<tr>
<td>Page 3</td>
<td>B{...}</td>
</tr>
<tr>
<td>Page 4</td>
<td>A{...}</td>
</tr>
<tr>
<td></td>
<td>B{...}</td>
</tr>
</tbody>
</table>
Predicates

Sargable predicates (**Search ARGuments-able**)
- Predicates that can be filtered by the RSS
- I.e., **column comparison-operator value**
- Where clause of query is put in Conjunctive Normal Form (CNF): term AND term AND term
- Each term is called a **boolean factor**
Predicates

Sargable predicates (Search ARGuments-able)
- Predicates that can be filtered by the RSS
- I.e., column comparison-operator value
- Where clause of query is put in Conjunctive Normal Form (CNF): term AND term AND term
- Each term is called a boolean factor

Examples of non-sargable
- function(column) = something
- column1 + column2 = something
- column + value = something
- column1 > column2
Predicates

Sargable predicates (Search ARGuments-able)
- Predicates that can be filtered by the RSS
- I.e., column comparison-operator value
- Where clause of query is put in Conjunctive Normal Form (CNF): term AND term AND term
- Each term is called a boolean factor

A predicate matches an index if
1. Predicate is sargable
2. Columns referenced in the predicate match an initial subset of attributes of the index key

Example: B-tree Index on (name, age)

- predicate1: name='xxx' and age='17'  match
- predicate2: age='17'  not match
Computation cost: RSICARD

Calculate the selectivity factor $F$ for each boolean factor/predicate
Calculate the selectivity factor $F$ for each boolean factor/predicate

- If index exists
  \[ F = \frac{1}{\text{ICARD}(\text{index})} \]  # distinct keys
- else
  \[ F = \frac{1}{10} \]
Calculate the selectivity factor $F$ for each boolean factor/predicate

**column = value**
- If index exists, $F = 1/\text{ICARD}(\text{index})$  # distinct keys
- else, $1/10$

**column1 = column2**
- $1/\text{Max}(\text{ICARD}(\text{column1 index}), \text{ICARD}(\text{column2 index}))$
Computation cost: RSICARD

Calculate the selectivity factor F for each boolean factor/predicate

column = value
- If index exists  \( F = 1/\text{ICARD(index)} \)  # distinct keys
- else  \( 1/10 \)

column1 = column2
- \( 1/\text{Max(\text{ICARD(column1 index)}, \text{ICARD(column2 index)})} \)

Assumes each key value in the index with the smaller cardinality has a matching value in the other index

For each record in relation 1, \( \frac{\text{NCARD2}}{\text{ICARD2}} \) tuples in relation 2 will satisfy the predicate

Total number of selected tuples = \( \frac{\text{NCARD2} \times \text{NCARD1}}{\text{ICARD2}} \)

\( F = \frac{1}{\text{ICARD2}} \)
Computation cost: RSICARD

Calculate the selectivity factor $F$ for each boolean factor/predicate

**column = value**
- If index exists $F = 1/\text{ICARD}(\text{index})$ # distinct keys
- else $1/10$

**column1 = column2**
- $1 / \text{Max}(\text{ICARD}(\text{column1 index}), \text{ICARD}(\text{column2 index}))$

**column > value**
- $F = (\text{high key value} - \text{value}) / (\text{high key value} - \text{low key value})$
Calculate the selectivity factor $F$ for each boolean factor/predicate

- **Column = value**
  - If index exists: $F = 1/ICARD(index)$  # distinct keys
  - Else: $1/10$

- **Column1 = Column2**
  - $1 / \text{Max}(ICARD(column1 \text{ index}), ICARD(column2 \text{ index}))$

- **Column > value**
  - $F = \text{(high key value} - \text{value}) / \text{(high key value} - \text{low key value})$

- **Pred1 and Pred2**
  - $F = F(\text{pred1}) \times F(\text{pred2})$

- **Pred1 or Pred2**
  - $F = F(\text{pred1}) + F(\text{pred2}) - F(\text{pred1}) \times F(\text{pred2})$

- **Not pred**
  - $F = 1 - F(\text{pred})$
IO cost

Calculate the number of pages access through IO
IO cost

Calculate the number of pages access through IO

**segment scan**

- \( \text{IO} = \frac{\text{TCARD}(T)}{P} \)  # segment pages
Calculate the number of pages access through IO

**segment scan**
- IO = TCARD(T)/P  # segment pages

**unique index matching** (e.g., EMP.ID = ‘123’)
- IO = 1 data page + 1-3 index page
IO cost

Calculate the number of pages access through IO

**segment scan**
- IO = TCARD(T)/P \# segment pages

**unique index matching** (e.g., EMP.ID = ‘123’)
- IO = 1 data page + 1-3 index page

**clustered index matching**
- IO = F(preds) * (NINDEX(I) + TCARD(T)) \# index pages & # data pages
Calculate the number of pages access through IO

**segment scan**
- $IO = \frac{T\text{CARD}(T)}{P}$  
  \# segment pages

**unique index matching** (e.g., EMP.ID = '123')
- $IO = 1$ data page + 1-3 index page

**clustered index matching**
- $IO = F(\text{preds}) \times (N\text{INDEX}(I) + T\text{CARD}(T))$  
  \# index pages & \# data pages

**non-clustered index matching**
- $IO = F(\text{preds}) \times (N\text{INDEX}(I) + N\text{CARD}(T))$  
  \# index pages & \# data page accesses
Calculate the number of pages access through IO

**segment scan**
- \( IO = \frac{TCARD(T)}{P} \) \# segment pages

**unique index matching** (e.g., EMP.ID = ‘123’)
- \( IO = 1 \) data page + 1-3 index page

**clustered index matching**
- \( IO = F(preds) \times (NINDEX(I) + TCARD(T)) \) \# index pages & \# data pages

**non-clustered index matching**
- \( IO = F(preds) \times (NINDEX(I) + NCARD(T)) \) \# index pages & \# data page accesses

**clustered index no matching**
- \( IO = NINDEX(I) + TCARD(T) \)
Final Cost

Cost = IO cost + Computation cost

= #I/Os + W * RSICARD
Access Path Selection for Joins

$R \bowtie S$

**Method 1: nested loops**
- Tuple order within a relation does not matter

**Method 2: merging scans**
- Both relations sorted on the join key
Access Path Selection for Joins

R \bowtie S

Method 1: nested loops
• Tuple order within a relation does not matter

Method 2: merging scans
• Both relations sorted on the join key

Tuple order is an interesting order if specified by
• Group by
• Order by
• Equi-join key

Search space too large!
Search Space – Join Order

Convention: right child is the inner relation
Convention: *right child is the inner relation*

For nested-loop join or hash join, a left-deep tree allows tuples to be passed through pipelining.
Search Space – Join Order

Convention: **right child is the inner relation**

For nested-loop join or hash join, a left-deep tree allows tuples to be passed through pipelining
Convention: **right child is the inner relation**

For nested-loop join or hash join, a left-deep tree allows tuples to be passed through pipelining

Bushy tree may produce cheaper plans but are rarely considered due to the explosion of search space
Partial group by can also reduce cost

Example:

```sql
SELECT D.name, count(*)
FROM EMP as E, DEPT as D
WHERE E.DeptID = D.DeptID
GROUP BY D.name
```

E has 10000 tuples
D has 100 tuples
Partial group by can also reduce cost

Example:

```sql
SELECT D.name, count(*)
FROM EMP as E, DEPT as D
WHERE E.DeptID = D.DeptID
GROUP BY D.name
```

E has 10000 tuples
D has 100 tuples
Modern query optimizers consider storage hierarchy and multicore?
Retain fast performance in the huge search space?
• Multi-user, multi-core, multi-tier storage, different operators, etc.

What about distributed system?
Modify query plans in the middle of execution?
Consider the following schema and SQL query

Relation R (a, b): 10 million tuples, R.a is the primary key
Relation S (c, d): 100 million tuples, S.c is a foreign key referring to R.a

```
SELECT *
FROM R, S WHERE R.a = S.c AND R.b = 5;
```

c) [10 points] Please estimate the number of rows in the output relation using the techniques in Selinger’79 (Lecture 6, Query Optimization).
Submit review for