CS 764: Topics in Database Management Systems
Lecture 4: Query Optimization

Xiangyao Yu
9/16/2020
Consider a nested loop join between R and S. Initially R and S are both stored on disk. The buffer management policy is DBMIN.

- | R | = 4
- | S | = 10
- | M | = 6

Q1: How many pages need to be read from disk to perform the join?

  4 pages to load R (locality set = 4)
  + 10 pages to load S (locality set = 1)
Consider a nested loop join between R and S. Initially R and S are both stored on disk. The buffer management policy is DBMIN.

- |R| = 4
- |S| = 10
- |M| = 4

Q2: Does the answer to Q1 change when |M| = 4? What is the buffer management policy for R and S in this case?

R: locality set = 3 pages
S: locality set = 1 page
Load S: 10 pages from disk
Load R + misses due to replacement: 3 + 10 = 13 pages from disk
Access Path Selection
in a Relational Database Management System

P. Griffiths Selinger
M. M. Astrahan
D. D. Chamberlin
R. A. Lorie
T. G. Price

IBM Research Division, San Jose, California 95193

ABSTRACT: In a high level query and data manipulation language such as SQL, requests are stated non-procedurally, without reference to access paths. This paper describes how System R chooses access paths for both simple (single relation) and complex queries (such as joins), given a user specification of desired data as a boolean expression of predicates. System R is an experimental database management system developed to carry out research on the relational model of data. System R was designed and built by members of the IBM San Jose Research Laboratory.

1. Introduction

System R is an experimental database management system based on the relational model of data which has been under development at the IBM San Jose Research Laboratory since 1975 [1]. The software was developed as a research vehicle in relational database, and is not generally available outside the IBM Research Divi-
Agenda

Query Optimization: Motivation

Query Optimization in R
  • Notation
  • Cost of single relation access paths
  • Access path selection for Join
  • Nest Queries
  • Limitations
Query Optimization: Motivation
Example SQL Query

SELECT *
FROM A, B, C
WHERE A.x = B.x
AND B.y = C.y
AND A.z = 13
AND B.y > 90
AND C.x < ‘XYZ’

How to evaluate this query?
Example SQL Query

SELECT *  
FROM A, B, C  
WHERE A.x = B.x  
AND B.y = C.y  
AND A.z = 13  
AND B.y > 90  
AND C.x < 'XYZ'

How to evaluate this query?

Solution 1:
  cross-product
  -> discard tuples based on predicates

This solution is too expensive
Example SQL Query

SELECT *
FROM A, B, C
WHERE A.x = B.x
AND B.y = C.y
AND A.z = 13
AND B.y > 90
AND C.x < 'XYZ'

How to evaluate this query?

Solution 2:
Example SQL Query

SELECT *
FROM A, B, C
WHERE A.x = B.x
AND B.y = C.y
AND A.z = 13
AND B.y > 90
AND C.x < ‘XYZ’

How to evaluate this query?

Solution 2:

Solution 3:
A query can be executed in multiple ways
Query optimizer goal: **SQL -> optimized execution plan**
Key decisions: (1) single relation access plan (2) join order
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
### Statistics

<table>
<thead>
<tr>
<th><strong>NCARD(T)</strong></th>
<th># tuples in T</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TCARD(T)</strong></td>
<td># of pages containing tuples in T</td>
</tr>
<tr>
<td><strong>P(T)</strong></td>
<td>Fraction of segment pages that hold tuples of T. P(T) = TCARD(T) / # non-empty pages in the segment</td>
</tr>
<tr>
<td><strong>ICARD(I)</strong></td>
<td># distinct keys in the index I</td>
</tr>
<tr>
<td><strong>NINDEX(I)</strong></td>
<td># pages in index I</td>
</tr>
<tr>
<td><strong>High key value and low key value</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Modern systems</strong></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: 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

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

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

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

**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**

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

Calculate the number of pages access through IO

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

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

Calculate the number of pages access through IO

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

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

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

**non-clustered index matching**

- IO = F(preds) * (NINDEX(I) + NCARD(T))  # index pages & # data page accesses
IO cost

Calculate the number of pages access through IO

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

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

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

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

**clustered index no matching**
- \( \text{IO} = \text{NINDEX}(I) + \text{TCARD}(T) \)
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

More on join cost in the next lecture
Access Path Selection for Joins – Example

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

Index on EMP.DNO, DEPT.DNO, EMP.JOB, JOB.JOB

Interesting order: (1) DNO, (2) JOB
Access Paths for Each Relation

Access plans for EMP:

• unordered
  • Segment scan
• DNO order
  • Segment scan + sort
  • JOB index scan + sort
  • DNO index scan
• JOB order
  • Segment scan + sort
  • JOB index scan
  • DNO index scan + sort
Access Paths for Each Relation

Access plans for EMP:
• unordered
• DNO order
• JOB order

Access plans for DEPT
• unordered
• DNO order

Access plans for JOB
• unordered
• JOB order
## Joining Relations

<table>
<thead>
<tr>
<th>JOB</th>
<th>⋈</th>
<th>EMP</th>
<th>⋈</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 access plans</td>
<td>3 access plans</td>
<td>2 access plans</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Join(JOB, EMP): \(3 \times 2\)

- Access plans
Joining Relations

\[
\text{JOB} \bowtie \text{EMP} \bowtie \text{DEPT}
\]

2 access plans 3 access plans 2 access plans

Join(JOB, EMP): \(3 \times 2 \times 2\)
- Access plans
- Join methods: nested-loop vs. merging scan
Joining Relations

\[ \text{JOB} \bowtie \text{EMP} \bowtie \text{DEPT} \]

2 access plans  3 access plans  2 access plans

Join(JOB, EMP): $3 \times 2 \times 2 \times 2$

- Access plans
- Join methods: nested-loop vs. merging scan
- Join order: inner vs. outer
Joining Relations

\[ \text{JOB} \bowtie \text{EMP} \bowtie \text{DEPT} \]

2 access plans  3 access plans  2 access plans

Join(JOB, EMP): \(3 \times 2 \times 2 \times 2 = 24\)
- Access plans
- Join methods: nested-loop vs. merging scan
- Join order: inner vs. outer

Join(EMP, DEPT): \(3 \times 2 \times 2 \times 2 = 24\)
- Access plans
- Join methods
- Join order
Joining Relations

\[ \text{JOB} \bowtie \text{EMP} \bowtie \text{DEPT} \]

- 2 access plans
- Join methods: nested-loop vs. merging scan
- Join order: inner vs. outer

Join(JOB, EMP): \(3 \times 2 \times 2 \times 2 = 24\)
- Access plans
- Join methods: nested-loop vs. merging scan
- Join order: inner vs. outer

Join(Join(JOB, EMP), DEPT)
Join(JOB, Join(EMP, DEPT))

Join(EMP, DEPT): \(3 \times 2 \times 2 \times 2 = 24\)
- Access plans
- Join methods
- Join order

Join(EMP, DEPT)
Joining Relations

Join( JOB, EMP ) ⋈ DEPT

2 access plans

Join( JOB, EMP ): 3 \times 2 \times 2 \times 2 = 24

- Access plans
- Join methods: nested-loop vs. merging scan
- Join order: inner vs. outer

Join( JOB, Join(EMP, DEPT) )

Many of these plans can be pruned early
(More on this next lecture)
Nested Queries

select name from emp where salary > (select avg(salary) from emp);

• Optimize and compute the inner block before evaluating the outer block
Nested Queries

```
select name from emp where salary >
    (select avg(salary) from emp);
```

- Optimize and compute the inner block before evaluating the outer block

```
select name from emp E where salary >
    (select salary from emp M where M.ID=E.mgrID)
```

- Subquery evaluated once for every emp tuple in the outer query block! This is very expensive
Nested Queries

select name from emp where salary >
(select avg(salary) from emp);
• Optimize and compute the inner block before evaluating the outer block

select name from emp E where salary >
(select salary from emp M where M.ID=E.mgrID)
• Subquery evaluated once for every emp tuple in the outer query block! This is very expensive

Alternatively

<table>
<thead>
<tr>
<th>SELECT</th>
<th>E.name</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM</td>
<td>emp E, emp M</td>
</tr>
<tr>
<td>WHERE</td>
<td>E.salary &gt; M.salary</td>
</tr>
<tr>
<td>AND</td>
<td>M.ID=E.mgrID</td>
</tr>
</tbody>
</table>
Nested Queries

```
select name from emp where salary >
  (select avg(salary) from emp);
  • Optimize and compute the inner block before evaluating the outer block

select name from emp E where salary >
  (select salary from emp M where M.ID=E.mgrID)
  • Subquery evaluated once for every emp tuple in the outer query block! This is very expensive
```

Alternatively

```
SELECT E.name
FROM emp E, emp M
WHERE E.salary > M.salary
AND M.ID=E.mgrID
```

Is this predicate sargable?
Limitations

• Optimizer complexity: $O(n2^{n-1})$, $n$ is the number of tables
• Ignore group by and aggregates optimizations
• Limited optimization of nested queries
• Cost model too simplistic
• RSS allows tuples from different relations on the same page; modern systems don’t do this
Experimental validation of cost functions?
  • More accurate cost functions can be used for specific systems

How to prune access paths? (more on this next lecture)

Can segment scan be better than index scan?

What’s more common? Procedural vs. non-procedural?

Are queries CPU bound?

How is the weighting factor ($W$) determined?

Is the optimizer optimal?
Group Discussion

SELECT EMP
FROM EMP
WHERE DNAME = 'CS';

Q1: What are the possible access paths on EMP?
Q2: Assume selectivity factor F = 1/10 for predicate DNAME='CS', which access path should be picked for the query above?

<table>
<thead>
<tr>
<th>ENAME</th>
<th>DNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Before Next Lecture

Submit discussion summary to https://wisc-cs764-f20.hotcrp.com
- Title: Lecture 4 discussion. group ##
- Authors: Names of students who joined the discussion
- Summary submission Deadline: Thursday 11:59pm

Before next lecture, submit review for