WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

CS 764: Topics in Database Management Systems
Lecture 4: Query Optimization-1

Xiangyao Yu
9/16/2020

Discussion Highlights

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)

Discussion Highlights

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

Today’s Paper: Query Optimization-1

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 Division. Nor does a user specify in what order joins are to be performed. The System R optimizer chooses both join order and an access path for each table in the SQL statement. Of the many possible choices, the optimizer chooses the one which minimizes "total access cost" for performing the entire statement.

This paper will address the issues of access path selection for queries. Retrieval for data manipulation (UPDATE, DELETE) is treated similarly. Section 2 will describe the place of the optimizer in the processing of a SQL statement, and section 3 will describe the storage component access paths that are available on a single physically stored table. In section 4 the optimizer cost formulas are introduced for single table queries, and section 5 discusses the joining of two or more tables, and their corresponding costs. Nested queries (queries in predicates) are covered in section 6.

SIGMOD 1979

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

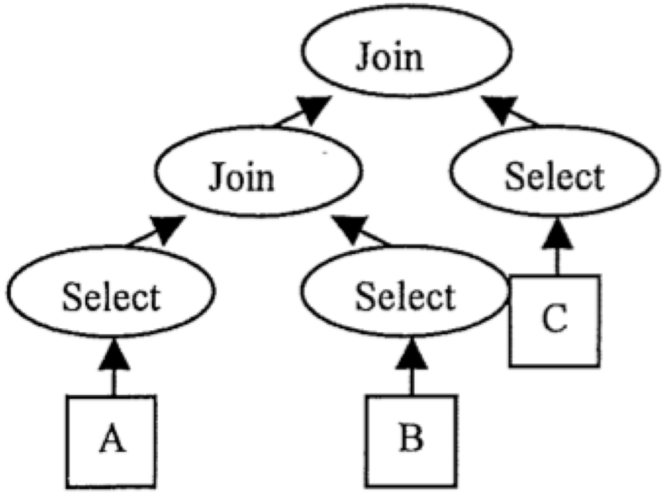
AND A.z = 13

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

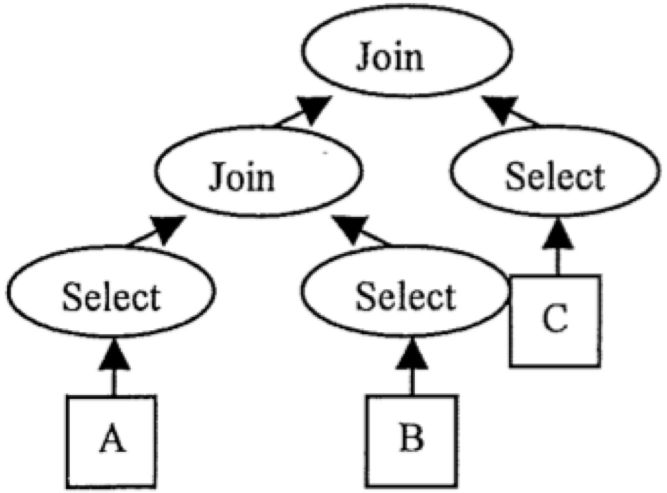
AND A.z = 13

AND B.y > 90

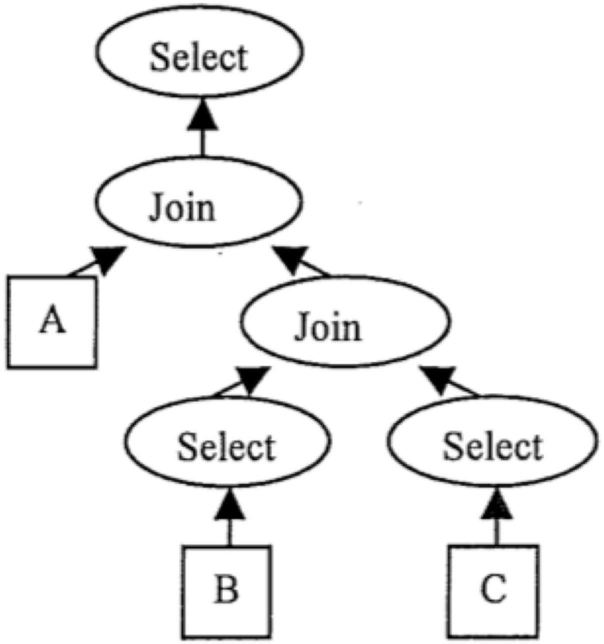
AND C.x < 'XYZ'

How to evaluate this query?

Solution 2:



Solution 3:



Example SQL Query

SELECT *

FROM A, B, C

WHERE A.x = B.x

AND   B.y = C.y

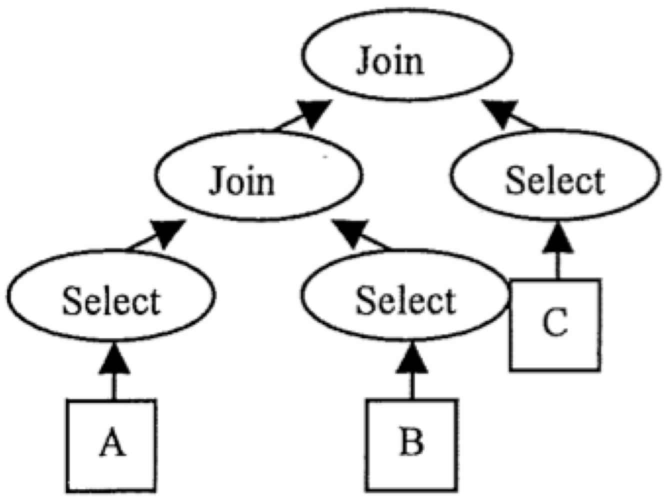
AND   A.z = 13

AND   B.y > 90

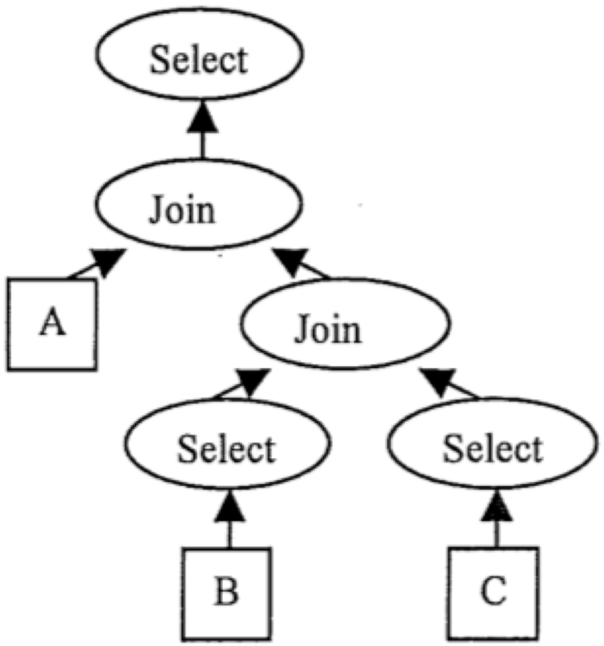
AND   C.x < ‘XYZ’

How to evaluate this query?

Solution 2:



Solution 3:



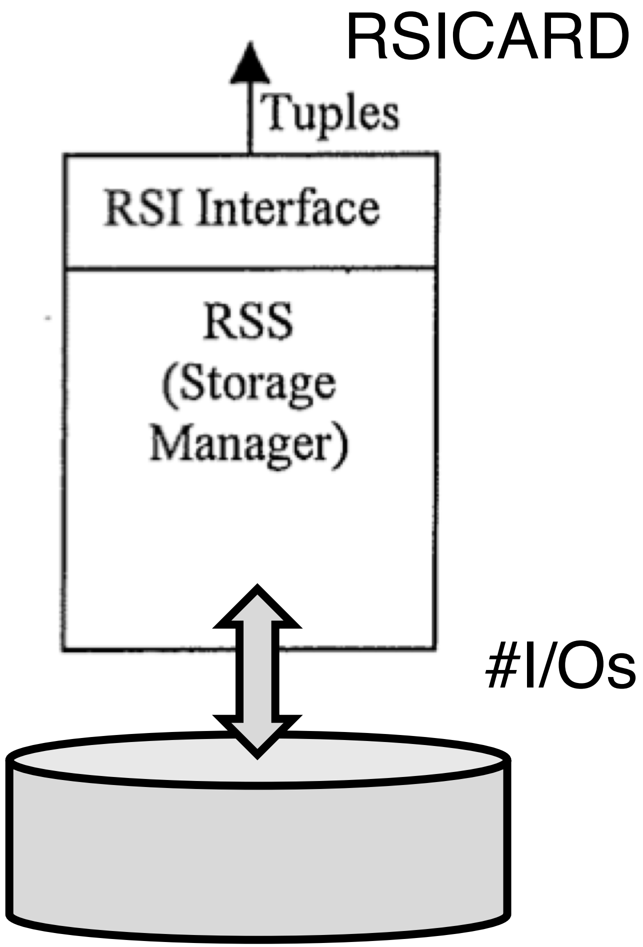
A query can be executed in multiple ways

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

NCARD(T)	# tuples in T
TCARD(T)	# of pages containing tuples in T
P(T)	Fraction of segment pages that hold tuples of T. P(T) = TCARD(T) / # non-empty pages in the segment
ICARD(I)	# distinct keys in the index I
NINDEX(I)	# pages in index I
High key value and low key value	
Modern systems	Keep histogram on table attributes.

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

Page 1	A{...} B{...}
Page 2	A{...}
Page 3	B{...}
Page 4	A{...} B{...}

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

Page 1	A{...} B{...}
Page 2	A{...}
Page 3	B{...}
Page 4	A{...} B{...}

Index Scan

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

Predicates

Sargable predicates (**S**earch **ARG**uments-**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 (**S**earch **ARG**uments-**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 (**S**earch **ARG**uments-**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

Computation cost: RSICARD

Calculate the selectivity factor F for each boolean factor/predicate

column = value

- If index exists F = 1/ICARD(index) # distinct keys
- else 1/10

Computation cost: RSICARD

Calculate the selectivity factor F for each boolean factor/predicate

column = value

- If index exists F = 1/ICARD(index) # distinct keys
- else 1/10

column > value

- F = (high key value - value) / (high key value – low key value)

Computation cost: RSICARD

Calculate the selectivity factor F for each boolean factor/predicate

column = value

- If index exists F = 1/ICARD(index) # distinct keys
- else 1/10

column > value

- F = (high key value - value) / (high key value – low key value)

pred1 and pred2

- F = F(pred1) * F(pred2)

pred1 or pred2

- F = F(pred1) + F(pred2) – F(pred1) * F(pred2)

Not pred

- F = 1– F(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

- 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

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

- 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

clustered index no matching

- IO = NINDEX(I) + TCARD(T)

Access Path Selection for Joins

R ⋈ 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 ⋈ 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

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

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

EMP

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

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

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

Joining Relations

JOB ⋈ **EMP** ⋈ **DEPT**

2 access plans | 3 access plans | 2 access plans

Join(JOB, EMP): **3×2**

- Access plans

Joining Relations

JOB $\bowtie$ **EMP** $\bowtie$ **DEPT**

2 access plans 3 access plans 2 access plans

Join(JOB, EMP): 3×2**×2**

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

Joining Relations

JOB ⋈ **EMP** ⋈ **DEPT**

2 access plans 3 access plans 2 access plans

Join(JOB, EMP): 3×2×2**×2**

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

Joining Relations

JOB $\bowtie$ **EMP** $\bowtie$ **DEPT**

2 access plans   3 access plans   2 access plans

Join(JOB, EMP): 3×2×2×2 = 24

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

Join(EMP, DEPT): 3×2×2×2 = 24

- Access plans
- Join methods
- Join order

Joining Relations

JOB $\bowtie$ **EMP** $\bowtie$ **DEPT**

2 access plans 3 access plans 2 access plans

Join(JOB, EMP): 3×2×2×2 = 24

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

Join(EMP, DEPT): 3×2×2×2 = 24

- Access plans
- Join methods
- Join order

Join(Join(JOB, EMP), DEPT)

Join(JOB, Join(EMP, DEPT))

Joining Relations

JOB ⋈ EMP ⋈ 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

Join(Join(JOB, EMP), DEPT)

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

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

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

Q/A – Query Optimization-1

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

EMP

ENAME	DNAME

TCARD = 100 # data pages
NCARD = TCARD * 100 # tuples
DEPT.IDX_ENAME (clustered)
DEPT.IDX_DNAME (non-clustered)

```
SELECT    ENAME
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?

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

- Surajit Chaudhuri, An Overview of Query Optimization in Relational Systems. PODS 1998.