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.

Today’s Paper: Query Optimization

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.

This paper assumes familiarity with relational data model terminology as described in Codd <7> and Date <8>. The user interface in System R is the unified query, data definition, and manipulation language SQL <5>. Statements in SQL can be issued both from an on-line casual-user-oriented terminal interface and from programming languages such as PL/I and COBOL.

In System R a user need not know how the tuples are physically stored and what access paths are available (e.g. which columns have indexes). SQL statements do not require the user to specify anything about the access path to be used for tuple retrieval. Nor does a user specify in what order joins are to be performed. The System R optimizer chooses both join order and an

Originally published in the *Proceedings of the 1979 ACM SIGMOD International Conference on the Management of Data*.

Digital recreation by Eric A. Brewer, brewer@cs.berkeley.edu, October 2002.

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.

2. Processing of an SQL statement

A SQL statement is subjected to four phases of processing. Depending on the origin and contents of the statement, these phases may be separated by arbitrary intervals of time. In System R these arbitrary time intervals are transparent to the system components which process a SQL statement. These mechanisms and a description of the processing of SQL statements from both programs and terminals are further discussed in <2>. Only an overview of those processing steps that are relevant to access path selection will be discussed here.

The four phases of statement processing are parsing, optimization, code generation, and execution. Each SQL statement is sent to the parser, where it is checked for correct syntax. A query block is represented by a SELECT list, a FROM list, and a WHERE tree, containing, respectively the list of items to be retrieved, the table(s) referenced, and the boolean combination of simple predicates specified by the user. A single SQL statement may have many query blocks because a predicate may have one operand which is itself a query.

If the parser returns without any errors detected, the OPTIMIZER component is called. The OPTIMIZER accumulates the names

23

SIGMOD 1979

Agenda

System R

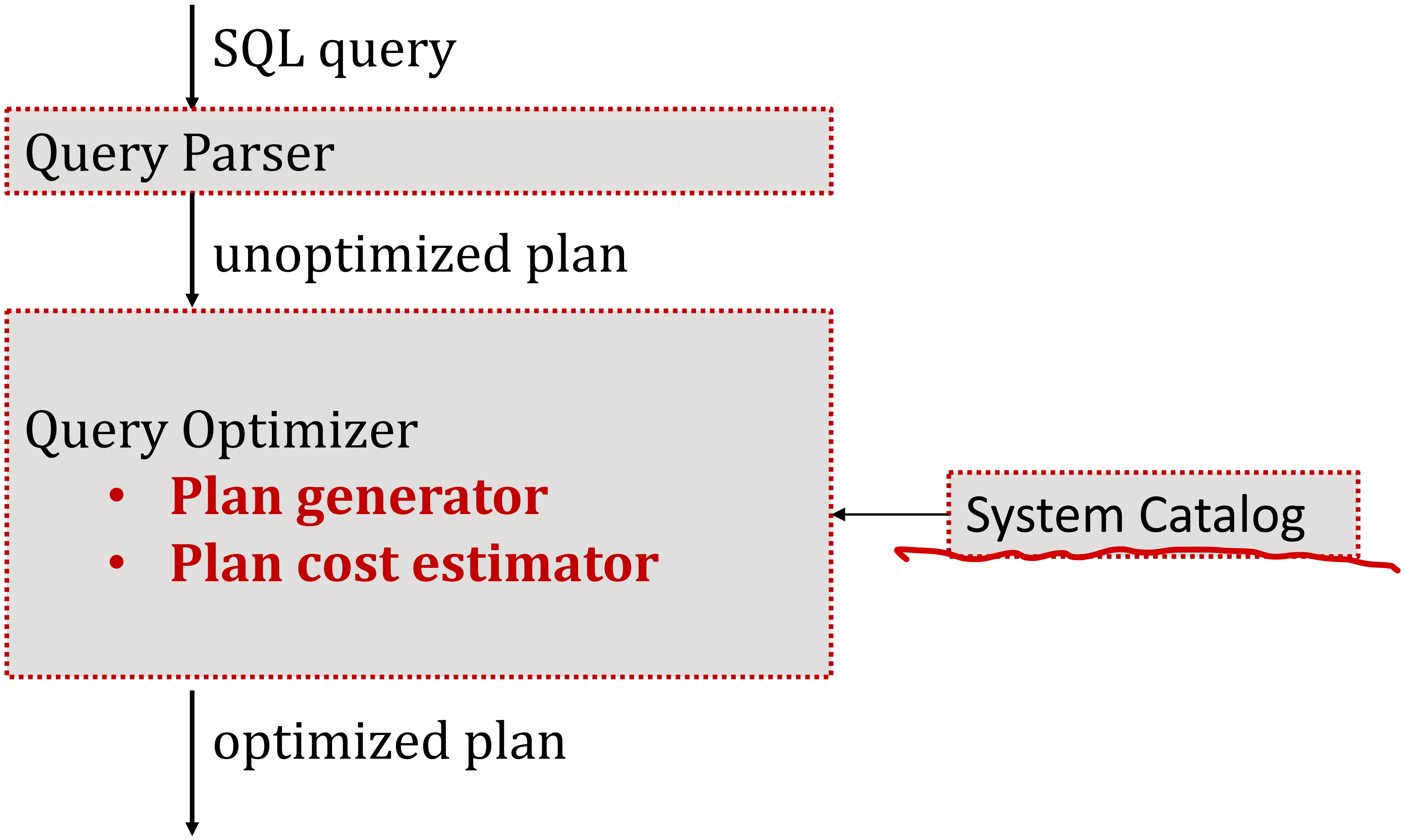
Query Optimization in R

- Cost estimation
- Plan enumeration

System R

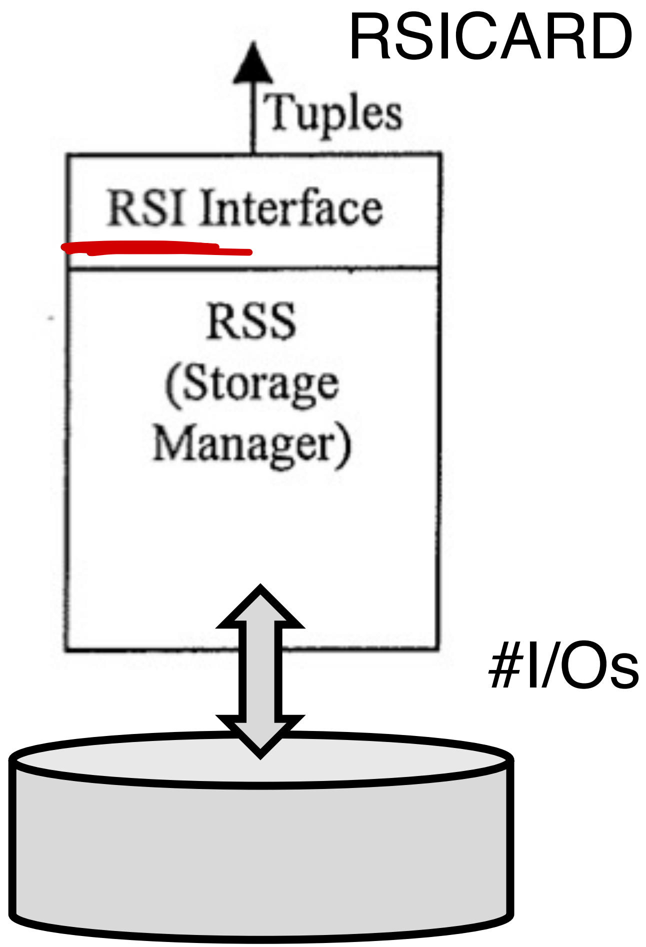
1. Parsing
2. Optimization
3. Code generation
4. Execution

Query Optimization



Query Optimization in System R

System R Storage Architecture

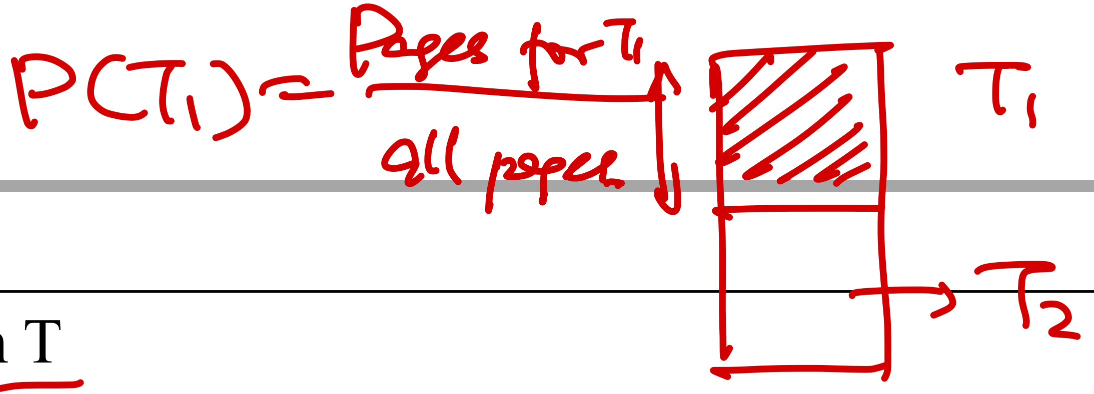


Cost = IO cost + Computation cost
= **#I/Os** + W * **RSICARD**

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{...}

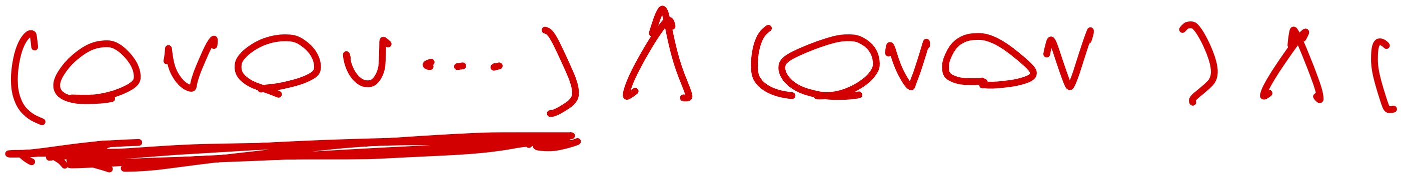
Index Scan

- 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**

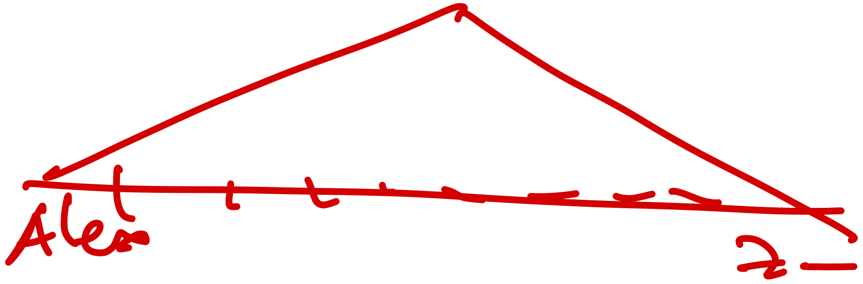
Examples of non-sargable

- 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: 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

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

column1 = column2

- 1 / Max(ICARD(column1 index), ICARD(column2 index))

Computation cost: RSICARD

Calculate the selectivity factor F for each boolean factor/predicate

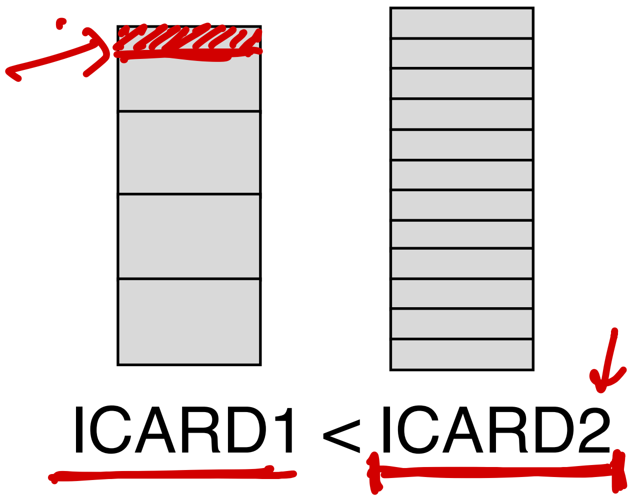
column = value

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

column1 = column2

- 1 / Max(ICARD(column1 index), 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, (NCARD2 / ICARD2) tuples in relation 2 will satisfy the predicate

Total number of selected tuples =

(NCARD2 * NCARD1) / ICARD2

F = 1 / ICARD2

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

column1 = column2

- 1 / Max(ICARD(column1 index), ICARD(column2 index))

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

column1 = column2

- 1 / Max(ICARD(column1 index), ICARD(column2 index))

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’)

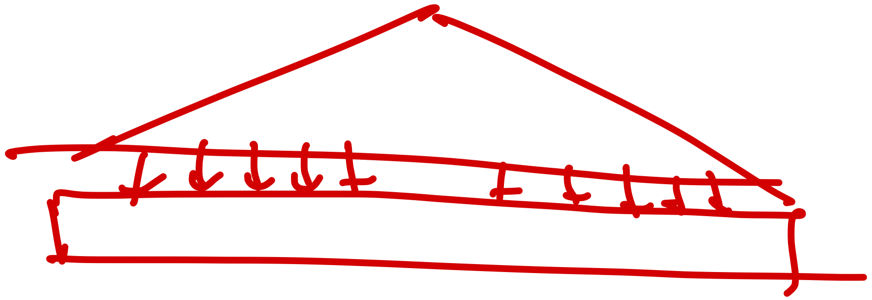
- IO = 1 data page + 1-3 index page

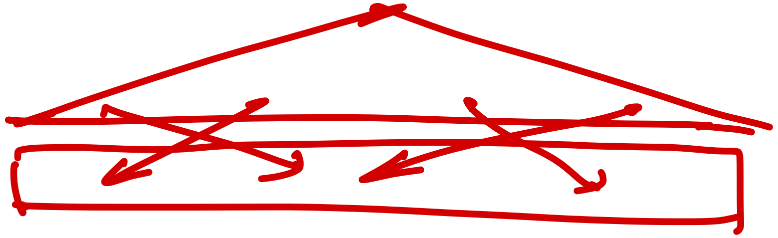
clustered index matching

- 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

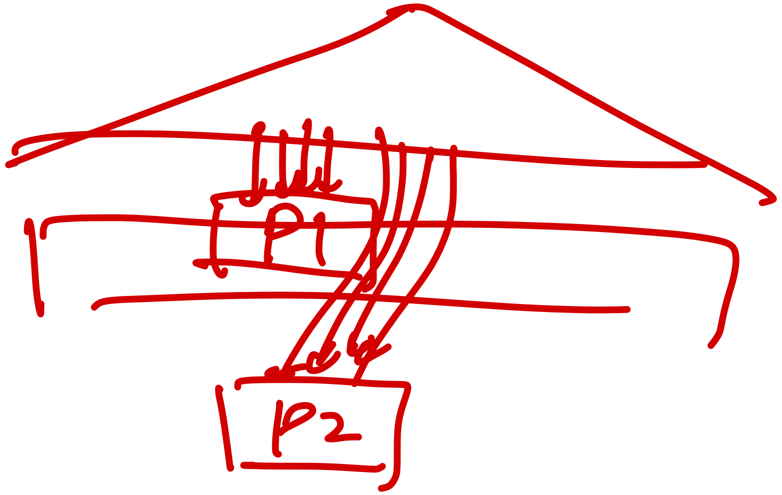
- IO = F(preds) * (NINDEX(I) + TCARD(T))   # index pages & # data pages

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

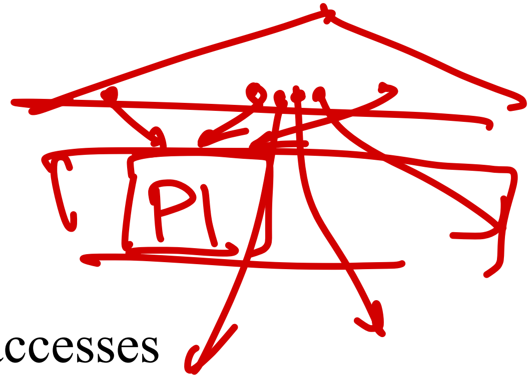
- IO = TCARD(T)/P # segment pages

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)

Final Cost

Cost = IO cost + Computation cost

= **#I/Os** + W * **RSICARD**

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

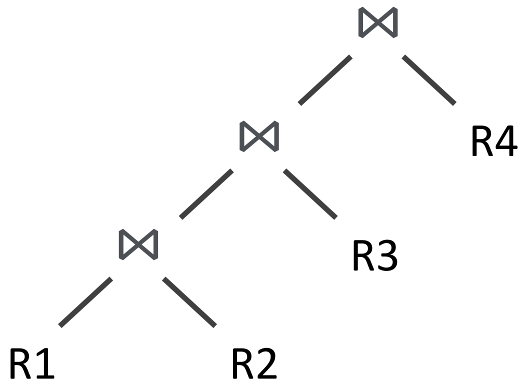
- Both relations sorted on the join key

Tuple order is an interesting order if specified by

- Group by
- Order by
- Equi-join key

Search space too large!

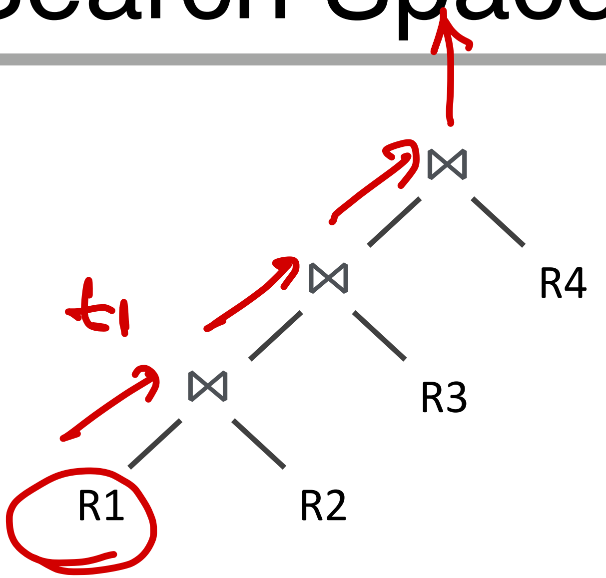
Search Space – Join Order



left-deep tree

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

Search Space – Join Order



left-deep tree

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

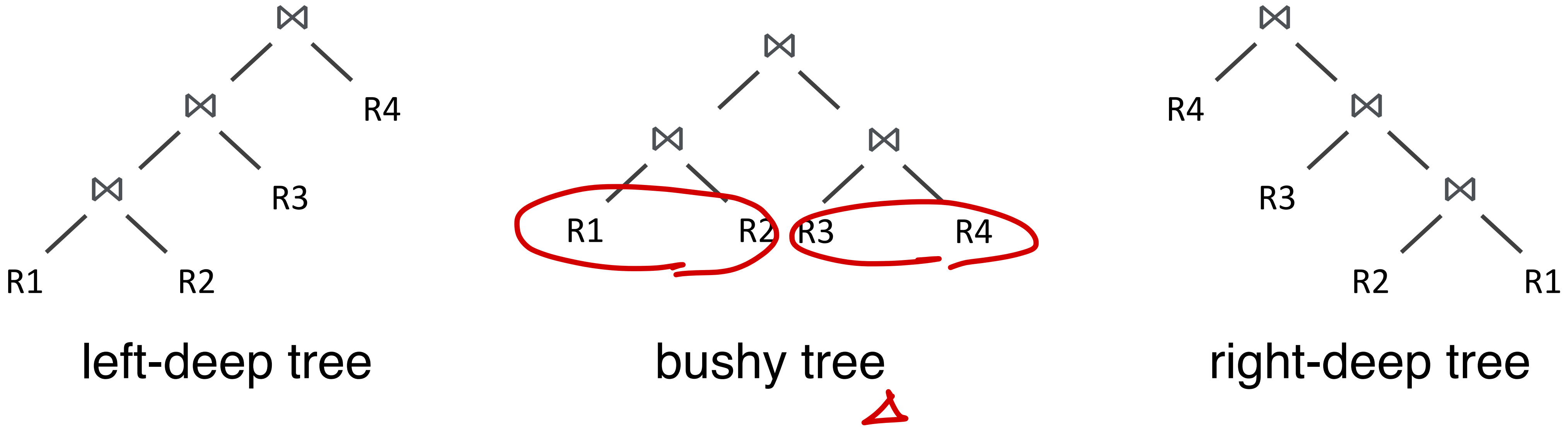
For nested-loop join or hash join, a left-deep tree allows tuples to be passed through pipelining

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

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

Bushy tree may produce cheaper plans but are rarely considered due to the explosion of search space

Search Space – Group By

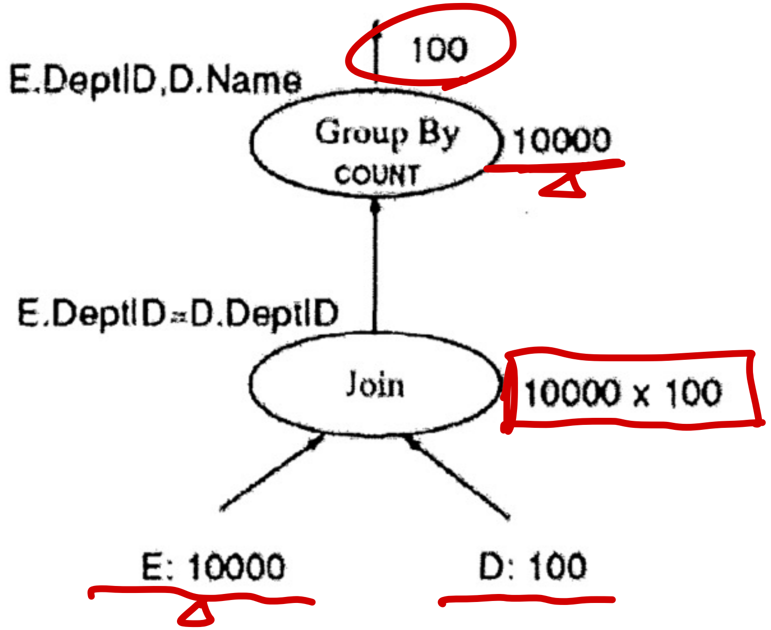
Partial group by can also reduce cost

Example:

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



Plan 1: Group by after join

Search Space – Group By

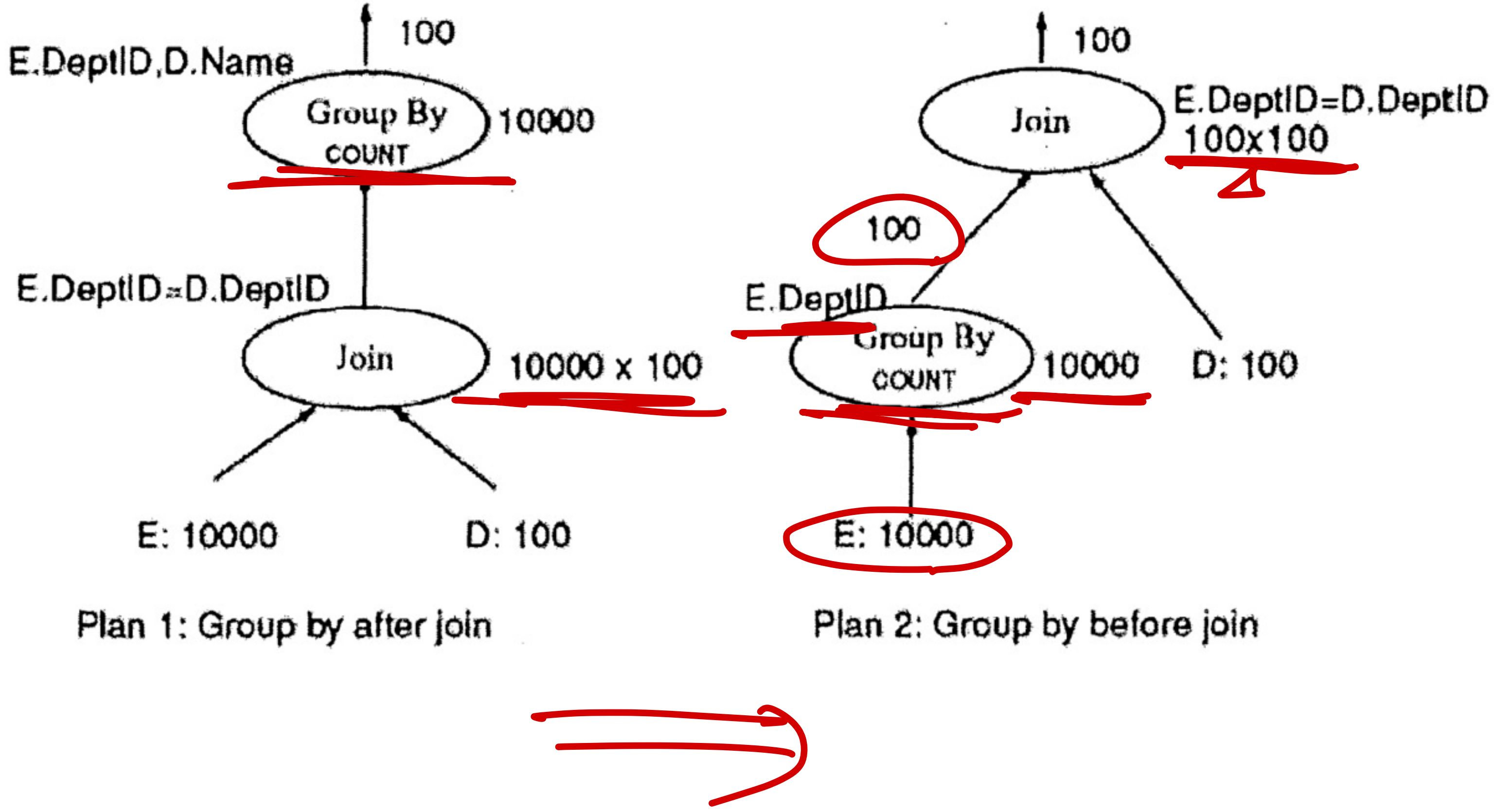
Partial group by can also reduce cost

Example:

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



Plan 1: Group by after join

Plan 2: Group by before join

Q/A – Query Optimization

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? →

Group Discussion

Q2.1 from CS764 Exam 2021

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.d = 5;
```

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

Before Next Lecture

Submit review for

Mike Stonebraker, et al. C-store: a column-oriented DBMS, VLDB 2005