PURDUE UNIVERSITY | Department of Computer Science

CS 44800: Introduction To Relational Database Systems

Query Optimization

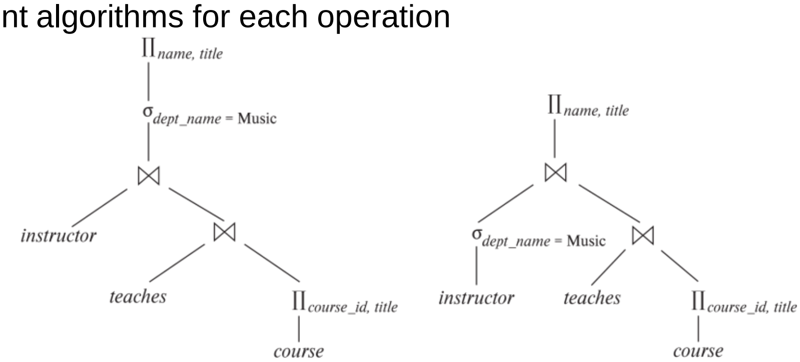
Prof. Chris Clifton

21 October 2021

Indiana Center for Database Systems™

Introduction

- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation



(a) Initial expression tree (b) Transformed expression tree

Relational algebra optimization

- Many ways to get the same result
  - Equivalent relational algebra expressions
  - Different algorithms for processing expressions
- Questions:
  - What are equivalent?
  - How do we determine what is best?
- Transformation rules
  - (preserve equivalence)
  - What are good transformations?

Rules: Natural joins & cross products & union

- $R \bowtie S = S \bowtie R$
- $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

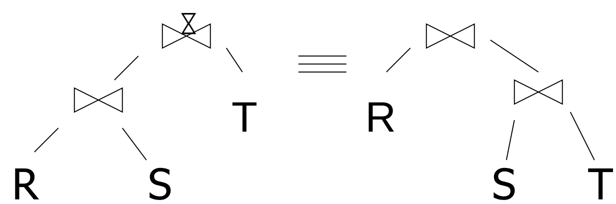
- R x S = S x R
- (R x S) x T = R x (S x T)

- R U S = S U R
- R U (S U T) = (R U S) U T

Note:

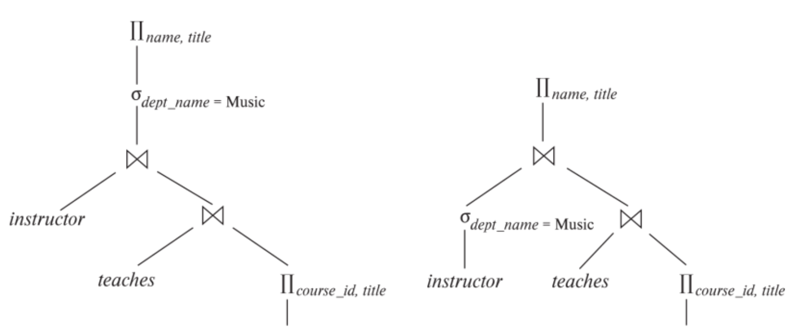
Department of Computer Science

- Carry attribute names in results, so order is not important
- Can also write as trees, e.g.:



Introduction

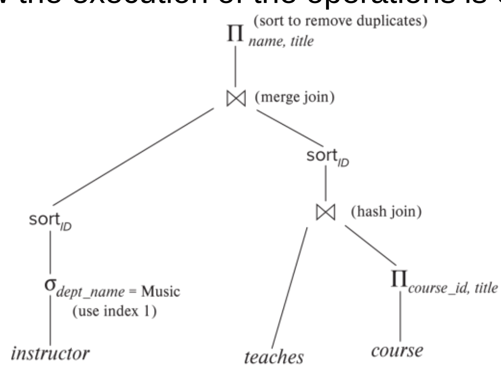
- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation



(a) Initial expression tree (b) Transformed expression tree

Introduction (Cont.)

- An **evaluation plan** defines exactly what algorithm is used for each operation, and how the execution of the operations is coordinated.



- Find out how to view query execution plans on your favorite database

Viewing Query Evaluation Plans

- Most database support **explain** <query>
  - Displays plan chosen by query optimizer, along with cost estimates
  - Some syntax variations between databases
    - Oracle: **explain plan for** <query> followed by **select** * **from** table (*dbms_xplan.display*)
    - SQL Server: **set showplan_text on**
- Some databases (e.g. PostgreSQL) support **explain analyse** <query>
  - Shows actual runtime statistics found by running the query, in addition to showing the plan
- Some databases (e.g. PostgreSQL) show cost as *f..l*
  - *f* is the cost of delivering first tuple and *l* is cost of delivering all results

PURDUE UNIVERSITY
Department of Computer Science

Optimization: Transform Query Plan

- Find the "tree" that gives the fastest response
  - All must give the same answer
  - Fewest IOs
- Rule-based Query Optimization
  - Transformations we know will always help
  - Independent of data values
- Cost-based Query Optimization
  - Estimate cost based on data

PURDUE UNIVERSITY
Department of Computer Science

Equivalent Query Plans

- Give the same set of tuples on EVERY legal database instance
  - Looking only at the schema
  - In practice, ignore integrity constraints
  - Note: since dealing with SQL, consider multiset semantics
- Equivalence Rule
  - Transformation that can be applied to a small set of operations as part of the larger tree
  - *Algebra…*

Rules: Selects

- $\sigma_{p1 \wedge p2}(R) = \quad \sigma_{p1}\ [\sigma_{p2}(R)]$
- $\sigma_{p1 \vee p2}(R) = \quad [\sigma_{p1}(R)] \cup [\sigma_{p2}(R)]$

Rules: Project

Let: X = set of attributes

Y = set of attributes

XY = X U Y

$$\pi_{xy}(R) = \cancel{\pi_x [\pi_y (R)]}$$

Multisets vs. Sets

- R = {a,a,b,b,b,c}
- S = {b,b,c,c,d}
- RUS = ?
  - Option 1   SUM
    RUS = {a,a,b,b,b,b,b,c,c,c,d}
  - Option 2   MAX
    RUS = {a,a,b,b,b,c,c,d}

Option 2 (MAX) makes this rule work:

$\sigma_{p1 \vee p2}(R) = \sigma_{p1}(R) \cup \sigma_{p2}(R)$

Example: R={a,a,b,b,b,c}

P1 satisfied by a,b; P2 satisfied by b,c

$\sigma_{p1 \vee p2}(R)$ = {a,a,b,b,b,c}

$\sigma_{p1}(R)$ = {a,a,b,b,b}

$\sigma_{p2}(R)$ = {b,b,b,c}

$\sigma_{p1}(R) \cup \sigma_{p2}(R)$ = {a,a,b,b,b,c}

“Sum” option makes more sense:

Senators (……)

Rep (……)

T1 = $\pi_{yr,state}$ Senators; T2 = $\pi_{yr,state}$ Reps

T1

Yr	State
97	CA
99	CA
98	AZ

T2

Yr	State
99	CA
99	CA
98	CA

Union?

Rules: σ + ⋈ combined

- Let p = predicate with only R attribs
  q = predicate with only S attribs
  m = predicate with only R,S attribs

- σ_p (R ⋈ S) = [σ_p (R)] ⋈ S
- σ_q (R ⋈ S) = R ⋈ [σ_q (S)]

Rules: σ + ⋈ combined (continued)

Some Rules can be Derived:

- $\sigma_{p \wedge q}$ (R ⋈ S) =
- $\sigma_{p \wedge q \wedge m}$ (R ⋈ S) =
- $\sigma_{p \vee q}$ (R ⋈ S) =

Derivation for first one

- $\sigma_{p \wedge q}$ (R ⋈ S) =
  - σ_p [σ_q (R ⋈ S)] =
  - σ_p [R ⋈ σ_q (S)] =
- [σ_p (R)] ⋈ [σ_q (S)]

Rules: Π, σ combined

- Let
  - x = subset of R attributes
  - z = attributes in predicate P (subset of R attributes)

$$\Pi_x[\sigma_p (R)] = \Pi_x\{\sigma_p [\overset{\Pi_{xz}}{\cancel{\Pi_x}} (R)]\}$$

PURDUE UNIVERSITY
Department of Computer Science

Rules: π, ⋈ combined

Let x = subset of R attributes

y = subset of S attributes

z = intersection of R,S attributes

$$\pi_{xy} (R \bowtie S) =$$

$$\pi_{xy}\{[\pi_{xz} (R)] \bowtie [\pi_{yz} (S)]\}$$

PURDUE UNIVERSITY

Department of Computer Science

$$\pi_{xy} \{\sigma_p \ (R \bowtie S)\} \ =$$

$$\pi_{xy} \{\sigma_p \ [\pi_{xz'}(R) \bowtie \pi_{yz'}(S)]\}$$

$$z' = z \cup \{\text{attributes used in P}\}$$

PURDUE UNIVERSITY

Department of Computer Science

Rules for σ, π combined with X

- similar...
- e.g., σ_p (R X S) = ?

PURDUE UNIVERSITY
Department of Computer Science

Rules: σ, U combined:

$\sigma_p(R \cup S) = \sigma_p(R) \cup \sigma_p(S)$

$\sigma_p(R - S) = \sigma_p(R) - S = \sigma_p(R) - \sigma_p(S)$

PURDUE UNIVERSITY
Department of Computer Science

Which are "good" transformations?

$\sigma_{p1 \wedge p2}(R) \rightarrow \sigma_{p1}[\sigma_{p2}(R)]$

$\sigma_p(R \bowtie S) \rightarrow [\sigma_p(R)] \bowtie S$

$R \bowtie S \rightarrow S \bowtie R$

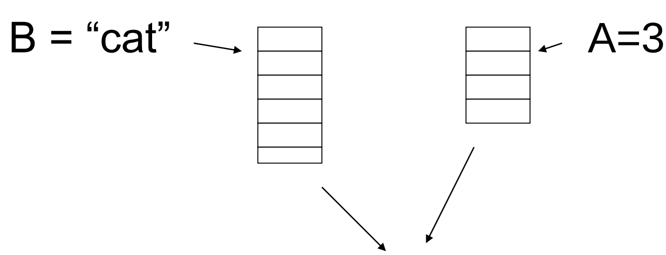
$\pi_x[\sigma_p(R)] \rightarrow \pi_x\{\sigma_p[\pi_{xz}(R)]\}$

Conventional wisdom: do projects early

- Example: R(A,B,C,D,E)   x={E}
  P: (A=3) $\wedge$ (B="cat")

- $\pi_x \{\sigma_p (R)\}$ vs. $\pi_E \{\sigma_p\{\pi_{ABE}(R)\}\}$

What if we have A, B indexes?



Intersect pointers to get
pointers to matching tuples

Bottom line:

- No transformation is always good
- Usually good: early selections
- More transformations:
  - Eliminate common sub-expressions
  - Other operations: duplicate elimination