Relational Algebra

Instructor: Shel Finkelstein

Reference:

A First Course in Database Systems,

3rd edition, Chapter 2.4 – 2.6, plus Query Execution Plans

Important Notices

- Lab3 assignment is due on **Sunday, May 19**, by 11:59pm.
  - 3 weeks to do Lab3 because of Midterm.
  - Lab2 was/will be discussed at Lab Sections.
  - Your solution should be submitted via Canvas as a zip file.
    - Canvas is used for both Lab submission and grading.
    - Late Lab Assignments will not be accepted.
    - Be sure that you post the correct file!
  - Load file for Lab3 has been posted to Piazza.
    - You must use load file to do Lab3, and original data must be used at multiple stages of Lab3.
    - Load data helps with testing, but we won’t post query solutions.

Important Notices

- **Reminder**: Midterm was on Wednesday, May 8.
  - Exam and Answers have been posted to Piazza under Resources→Exams
  - Answers were/will be discussed in class on Friday, May 10.
  - Midterms were returned during last 10 minutes of class on Monday, May 13.
  - If you have questions about the grading of the Midterm, please speak to your TA or to me soon!
- See Small Group Tutoring website for LSS Tutoring with Chandler Hawkins.

What is a Data Model?

- A *data model* is a mathematical formalism that consists of three parts:
  1. A notation for describing and representing data (<u>structure</u> of the data)
  2. A set of <u>operations</u> for manipulating data.
  3. A set of constraints on the data.

- What is the associated query language for the relational data model?

Two Query Languages

- Codd proposed two different query languages for the relational data model.
  - Relational Algebra
    - Queries are expressed as a sequence of operations on relations.
    - Procedural language.
  - Relational Calculus
    - Queries are expressed as formulas of first-order logic.
    - Declarative language.
- **Codd's Theorem**: The Relational Algebra query language has the same *expressive power* as the Relational Calculus query language.

Procedural vs. Declarative Languages

- **Procedural program**
  - The program is specified as a sequence of operations to obtain the desired the outcome. I.e., *how* the outcome is to be obtained.
  - E.g., Java, C, …

- **Declarative program**
  - The program specifies *what* is the expected outcome, and not *how* the outcome is to be obtained.
  - E.g., Scheme, Ocaml, …

SQL – Structured Query Language

- Is SQL a procedural or a declarative language?
  - SQL is usually described as declarative, but it's not fully declarative
  - However, relational database systems usually try to understand meaning of query, regardless of how query is expressed
    - There may be multiple equivalent ways to write a query

- SQL is the principal language used to describe and manipulate data stored in relational database systems.
  - Frequently pronounced as "Sequel", but formally it's "Ess Cue El"
  - Not the same as Codd's Relational Algebra or Relational Calculus

Some Properties of Good Database Query Languages and Database Systems

1. Physical database independence
   - Programmers should be able to write queries without understanding the mechanics of the physical layer
   - What was logical data independence?
2. Highly expressive
   - Programmers should be able to formulate simple and complex queries using the language.
3. Efficient execution
   - Systems should be able to compute answers to queries with "good" response time and throughput.

- Physical data independence is achieved by most query languages today.
- Increased expressiveness may come at the expense of not-so-good performance on some complex queries

Relational Algebra

- Relational Algebra: a query language for manipulating data in the relational data model.
  - Not used directly as a query language
- Internally, Relational Database Systems transform SQL queries into trees/graphs that are similar to relational algebra expressions.
  - Query analysis, transformation and optimization are performed based on these relational algebra expression-like representations.
  - Relational Databases use multi-sets/bags, but Relational Algebra is based on <u>sets</u>.
    - There are multi-set variations of Relational Algebra that permit duplicates, and that's more realistic for Relational Database …
    - … but we'll only discuss <u>set-based</u> Relational Algebra.

Composition

- Each Relational Algebra operator is either a unary or a binary operator.
- A complex Relational Algebra expression is built up from basic ones by composing simpler expressions.
- This is similar to SQL queries and views.

Relation Algebra Operators

- Queries in relational algebra are composed using basic operations or functions.
  - Selection (σ)
  - Projection (π)
  - Set-theoretic operations:
    - Union ($\cup$)
    - Set-difference (-)
    - Cross-product (x)
    - Intersection ($\cap$)
  - Renaming (ρ)
  - Natural Join ($\bowtie$), Theta-Join ($\bowtie_\theta$)
  - Division (/ or $\div$) [Not discussed in CMPS 182]

Relation Algebra Operators

- Codd proved that the relational algebra operators (σ , π , x , U , -) are independent of each other. That is, you can't define any of these operators using the others.

- However, there are other important operators that can be expressed using (σ , π , x , U , -)
  - Theta Join, Join, Natural Join, Semi-Join
  - Set Intersection
  - Division (/ or ÷) [Not discussed in CMPS 182]
  - Outer Join (sections 5.2.7 and 6.3.8), which we'll discuss when we get to OLAP, On-Line Analytic Processing (section 10.6)

Selection: $\sigma_{\textit{condition}}(R)$

- Unary operation
  - Input: Relation with schema $R(A_1, \ldots, A_n)$
  - Output: Relation with attributes $A_1, \ldots, A_n$
  - Meaning: Takes a relation R and extracts only the rows from R that satisfy the *condition*
  - Condition is a logical combination (using AND, OR, NOT) of expressions of the form:

    <expr> <op> <expr>

    where <expr> is an attribute name, a constant, a string, and op is one of (=, ≤, ≥, <, >, <>)

    - E.g., “age > 20 OR height < 6”,
    - “name LIKE ‘Anne%’ AND salary > 200000”
    - “NOT (age > 20 AND salary < 100000)”

Example of σ

- $\sigma_{rating > 6}$ (Hotels)

Hotels

name	address	rating	capacity
~~Windsor~~	~~54th ave~~	~~6.0~~	~~135~~
Astoria	5th ave	8.0	231
BestInn	45th st	6.7	28
~~ELodge~~	~~39 W st~~	~~5.6~~	~~45~~
~~ELodge~~	~~2nd E st~~	~~6.0~~	~~40~~

name	address	rating	capacity
Astoria	5th ave	8.0	231
BestInn	45th st	6.7	28

Example of σ with AND in Condition

- $\sigma_{\text{rating > 6 AND capacity > 50}}$ (Hotel)

name	address	rating	capacity
Windsor	54th ave	6.0	135
Astoria	5th ave	8.0	231
BestInn	45th st	6.7	28
ELodge	39 W st	5.6	45
ELodge	2nd E st	6.0	40

name	address	rating	capacity
Astoria	5th ave	8.0	231

- Is $\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C1 \text{ AND } C2}(R)$?
- Prove or give a counter-example.

- Is $\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C2}(\sigma_{C1}(R))$?
- Prove or give a counter-example.

Projection: $\pi_{\textit{<attribute list>}}(R)$

- Unary operation
  - Input : Relation with schema $R(A_1, \dots, A_n)$
  - Output: Relation with attributes in *attribute list,* which must be attributes of R
  - Meaning: For every tuple in relation R, output only the attributes appearing in *attribute list*
- May be duplicates; for Codd's Relational Algebra, duplicates are always eliminated (set-oriented semantics)
  - Reminder: For relational database, duplicates matter.
  - Why?

Example of π

- $\pi_{\text{name, address}}$ (Hotels)

name	address
Windsor	54th ave
Astoria	5th ave
BestInn	45th st
ELodge	39 W st
ELodge	2nd E st

- Suppose that name and address form the key of the Hotels relation. Is the cardinality of the output relation the same as the cardinality of Hotels? Why?

Example of π

- π_{name} (Hotel)

name
Windsor
Astoria
BestInn
ELodge

- Note that there are no duplicates.

Set Union: R U S

- Binary operator
  - Input: Two relations R and S which must be union-compatible
    - They have the same arity, i.e., the same number of columns.
    - For every column i, the i'th column of R has the same type as the i'th column of S.
    - Note that field names are not used in defining union-compatibility.
      - We can think of relations R and S as being union-compatible if they are sets of records having the same record type.
  - Output: Relation that has the same type as R (or same type as S).
  - Meaning: The output consists of the set of all tuples in either R or S (or both)

Example of U

Dell_Desktops U HP_Desktops

Dell_Desktops

Harddisk	Speed	OS
20G	500Mhz	Windows
30G	1.0Ghz	Windows
20G	750Mhz	Linux

HP_Desktops

Harddisk	Speed	OS
30G	1.2Ghz	Windows
20G	500Mhz	Windows

All tuples in R occurs in R U S.
All tuples in S occurs in R U S.
R U S contains tuples that either occur in R or S (or both).

Harddisk	Speed	OS
20G	500Mhz	Windows
30G	1.0Ghz	Windows
20G	750Mhz	Linux
30G	1.2Ghz	Windows

Properties of U

Dell_Desktops U HP_Desktops

RUS = SUR (commutativity)
(RUS)UT = RU(SUT) (associativity)

Dell_Desktops

Harddisk	Speed	OS
20G	500Mhz	Windows
30G	1.0Ghz	Windows
20G	750Mhz	Linux

HP_Desktops

Harddisk	Speed	OS
30G	1.2Ghz	Windows
20G	500Mhz	Windows

Harddisk	Speed	OS
20G	500Mhz	Windows
30G	1.0Ghz	Windows
20G	750Mhz	Linux
30G	1.2Ghz	Windows

Set Difference: R - S

- Binary operator.
  - Input: Two relations R and S which must be union-compatible
  - Output: Relation with the same type as R (or same type as S)

  - Meaning: Output consists of all tuples in R but not in S

Example of -

- Dell_Desktops - HP_Desktops

Dell_Desktops

Harddisk	Speed	OS
20G	500Mhz	Windows
30G	1.0Ghz	Windows
20G	750Mhz	Linux

HP_Desktops

Harddisk	Speed	OS
30G	1.2Ghz	Windows
20G	500Mhz	Windows

Dell_Desktops – HP_Desktops

Harddisk	Speed	OS
30G	1.0Ghz	Windows
20G	750Mhz	Linux

Properties of -

- HP_Desktops – Dell_Desktops

Dell_Desktops

Harddisk	Speed	OS
20G	500Mhz	Windows
30G	1.0Ghz	Windows
20G	750Mhz	Linux

HP_Desktops

Harddisk	Speed	OS
30G	1.2Ghz	Windows
20G	500Mhz	Windows

HP_Desktops – Dell_Desktops

Harddisk	Speed	OS
30G	1.2Ghz	Windows

Is it commutative?
Is it associative?

Product: R x S

- Binary operator
  - Input: Two relations R and S, where R has relation schema $R(A_1, \ldots, A_m)$ and S has relation schema $S(B_1, \ldots, B_n)$.
  - Output: Relation of arity m+n
  - Meaning:

    $R \times S = \{ (a_1, \ldots, a_m, b_1, \ldots, b_n) \mid (a_1, \ldots, a_m) \in R \text{ and } (b_1, \ldots, b_n) \in S) \}$.

    - Read "|" as "such that"
    - Read "∈" as "belongs to"

Example and Properties of Product

R

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

S

D	E
d_1	e_1
d_2	e_2
d_3	e_3

R x S

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_1	d_2	e_2
a_1	b_1	c_1	d_3	e_3
a_2	b_2	c_2	d_1	e_1
a_2	b_2	c_2	d_2	e_2
a_2	b_2	c_2	d_3	e_3

- Is x commutative?
- Is x associative?
- Is x distributive across ∪? That is, does Rx(S∪T) = (RxS) ∪ (RxT) ?

Product and Common Attributes

- What happens when we compute the Product of R and S if R and S contain common attributes, e.g., for R(A,B,C) and S(A,E)?

A.1	B	C	A.2	E
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_1	d_2	e_2
a_1	b_1	c_1	d_3	e_3
a_2	b_2	c_2	d_1	e_1
a_2	b_2	c_2	d_2	e_2
a_2	b_2	c_2	d_3	e_3

Derived Operators

- So far, we have learned:
  - Selection
  - Projection
  - Product
  - Union
  - Difference

- Some other operators can be derived by composing the operators we have learned so far:
  - Theta-Join, Join, Natural Join, Semi-Join
  - Set Intersection
  - Division/Quotient [Not discussed in CMPS 182]
  - Outer Join (to be discussed when we get to OLAP)

Theta-Join: $R \bowtie_{\theta} S$

- Binary operator
  - Input: $R(A_1, \ldots, A_m)$, $S(B_1, \ldots, B_n)$
  - Output: Relation consisting of all attributes $A_1, \ldots, A_m$ and all attributes $B_1, \ldots, B_n$. Identical attributes in R and S are disambiguated with the relation names.

  - Meaning of $R \bowtie_{\Theta} S$: The θ-Join outputs those tuples from R x S that satisfy the condition θ.
    - Compute R x S, then keep only those tuples in R x S that satisfy θ.
    - Equivalent to writing $\sigma_{\theta}(R \times S)$

- If θ always evaluates to TRUE, then $R \bowtie_{\Theta} S = \sigma_{\theta}(R \times S) = R \times S$.

Example of Theta-Join

Enrollment(esid, ecid, grade)

Course(cid, cname, instructor-name)

Please give me an example of a Theta-Join to write on the board where ecid in Enrollment equals cid in Course.

- Joins involving equality predicates (usually just called Joins or Equi-Joins) are very common in database; other joins are less common.
  - Enrollment $\bowtie_{\Theta}$ Course, where θ could be:
    "Enrollment.ecid = Course.cid"
- Could write <u>any</u> condition involving attributes of Enrollment and Course as θ, just as with σ.

Natural Join: R⋈S

- Often a query over two relations can be formulated using Natural Join.
- Binary operator:
  - Input: Two relations R and S where { A_1, …, A_k } is the set of common attributes (column names) between R and S.
  - Output: A relation where its attributes are attr(R) U attr(S). In other words, the attributes consists of the attributes in R x S <u>without repeats</u> of the common attributes { A_1, …, A_k }

- Meaning:
  $R \bowtie S = \pi_{(attr(R) \cup attr(S))} (\sigma_{R.A1=S.A1 \text{ AND } R.A2 = S.A2 \text{ AND } \dots \text{ AND } R.Ak=S.Ak} (R \times S))$

1. Compute R x S
2. Keep only those tuples in R x S satisfying:
   R.A1=S.A1 AND R.A2 = S.A2 AND … AND R.Ak=S.Ak
3. Output is projection on the set of attributes in R U S (without repeats of the attributes that appear in both)

Example of Natural Join

Enrollment(sid, cid, grade)
Course(cid, cname, instructor-name)

cid is the only common attribute appearing in both relations.

- Want the Natural Join of Enrollment and Course, which computes:
  StudentCourseGrade(sid, cid, grade, cname, instructor-name)

$$\pi_{\text{(sid, cid, grade, cname, instructor-name)}} (\sigma_{\text{Enrollment.cid=Course.cid}} (\text{Enrollment x Course}))$$

- What ‘s the Natural Join when R and S have <u>no</u> common attributes?
- What ‘s the Natural Join when R and S have <u>only</u> common attributes?

Semi-Join: $R \ltimes S$

- Meaning: $R \ltimes S = \pi_{attr(R)} (R \bowtie S)$

1. Compute Natural Join of R and S
2. Output the projection of that on just the attributes of R

- Find all courses that have some enrollment:
  Course $\ltimes$ Enrollment
- Find all faculty who are advising at least one student:
  Faculty $\ltimes$ Student

- How does Semi-Join relate to EXISTS in SQL?

Set Intersection: R∩S

Find all desktops sold by both Dell and HP.

Dell_Desktops ∩ HP_Desktops

Dell_Desktops

Harddisk	Speed	OS
20G	500Mhz	Windows
30G	1.0Ghz	Windows
20G	750Mhz	Linux

HP_Desktops

Harddisk	Speed	OS
30G	1.2Ghz	Windows
20G	500Mhz	Windows

Harddisk	Speed	OS
20G	500Mhz	Windows

Intersect

- How would you write Dell_desktops ∩ HP_desktops in SQL?

```
SELECT *
FROM Dell_desktops

INTERSECT

SELECT *
FROM HP_desktops;
```

- Intersection is a <u>Derived Operator</u> in Relational Algebra:

$$\begin{aligned} R \cap S &= R - (R - S) \\ &= S - (S - R) \end{aligned}$$

Independence of Basic Operators

- Many interesting queries can be expressed using the five basic operators (σ , π , x , U , -)
- Can one of the five operators be derived by the other four operators?

<u>Theorem (Codd):</u>

The five basic operators are independent of each other. In other words, for each relational operator *o*, there is no relational algebra expression that is built from the rest that defines *o*.

- x
- U
- π
- σ
- -

Renaming: $\rho_{S(A1, \ldots, An)}$ (R)

- To specify the attributes of a relational expression.
- Input: a relation, a relation symbol R, and a set of attributes {B1, … ,Bn}
- Output: the same relation with name S and attributes A1, …, An.

- Meaning: rename relation R to S with attributes A1, …, An.

- Example: $\rho_{\text{BeersInfo(beer,maker)}}$ Beers(name, manuf)

Example

R

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

S

C	D
d_1	e_1
d_2	e_2
d_3	e_3

$R \times \rho_{T(X,D)} S$

A	B	C	X	D
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_1	d_2	e_2
a_1	b_1	c_1	d_3	e_3
a_2	b_2	c_2	d_1	e_1
a_2	b_2	c_2	d_2	e_2
a_2	b_2	c_2	d_3	e_3

More Complex Queries

- Relational operators can be composed to form more complex queries. We have already seen examples of this in SQL.

Enrollments(<u>esid, ecid</u>, grade)

Courses(<u>cid</u>, cname, instructor-name)

- Query 1: Find the student id, grade and instructor where the student had a grade that was more than 80 points in a course.

$$\sigma_{\text{grade}>80} \left(\pi_{\text{esid, grade, instructor-name}} \left(\sigma_{\text{Enrollments.ecid = Courses.cid}} (\text{Enrollments x Courses}) \right) \right)$$

Query 2

Enrollments(<u>esid, ecid</u>, grade)

Courses(<u>cid</u>, cname, instructor-name)

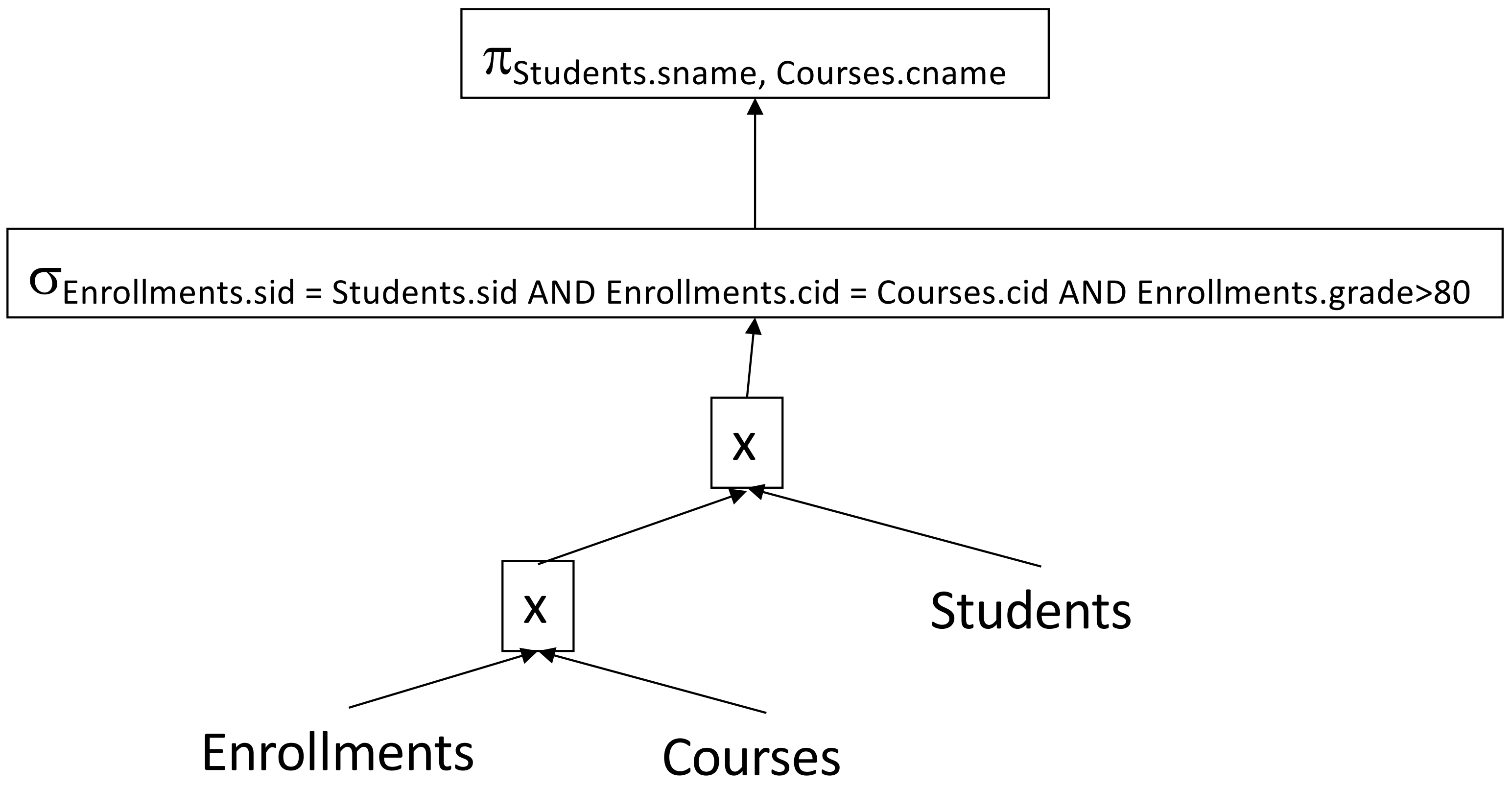
Students(<u>sid</u>, sname)

- Find the student name and course name where the student had a grade that was more than 80 points in a course.

$$\pi_{\text{Students.sname, Courses.cname}} \left(\sigma_{\substack{\text{Enrollments.ecid = Courses.cid} \\ \text{AND Enrollments.esid = Students.sid} \\ \text{AND Enrollments.grade > 80}}} (\text{Enrollments x Courses x Students}) \right)$$

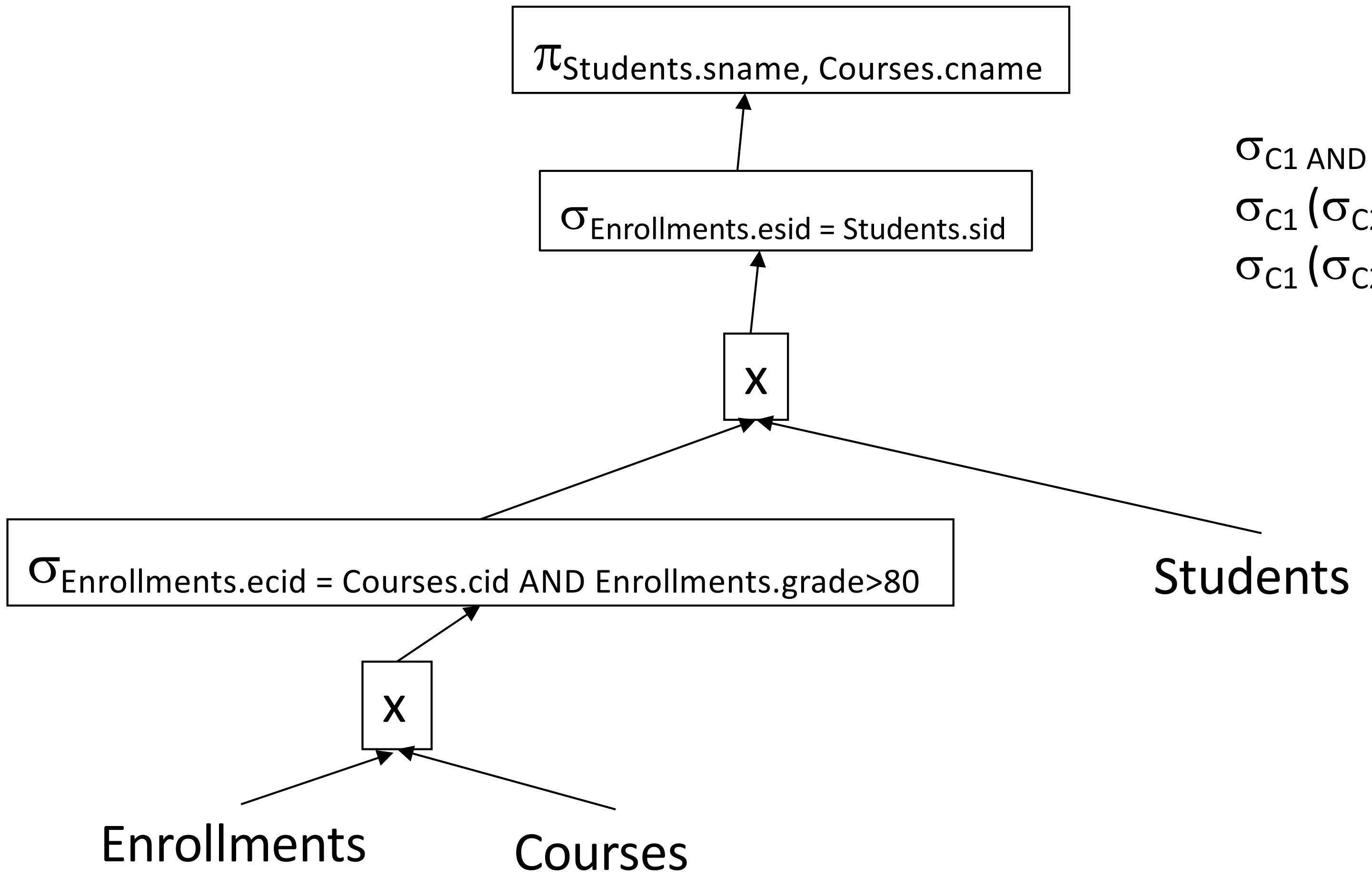
An Execution Plan for Query 2

- Find the student name and course name where the student had a grade more than 80 points in a course.


$\pi_{\text{Students.sname, Courses.cname}}$
$\sigma_{\text{Enrollments.sid = Students.sid AND Enrollments.cid = Courses.cid AND Enrollments.grade>80}}$
x
x
Students
Enrollments
Courses

<u>Another</u> Execution Plan for Query 2

- Find the student name and course name where the student had a grade more than 80 points in a course.

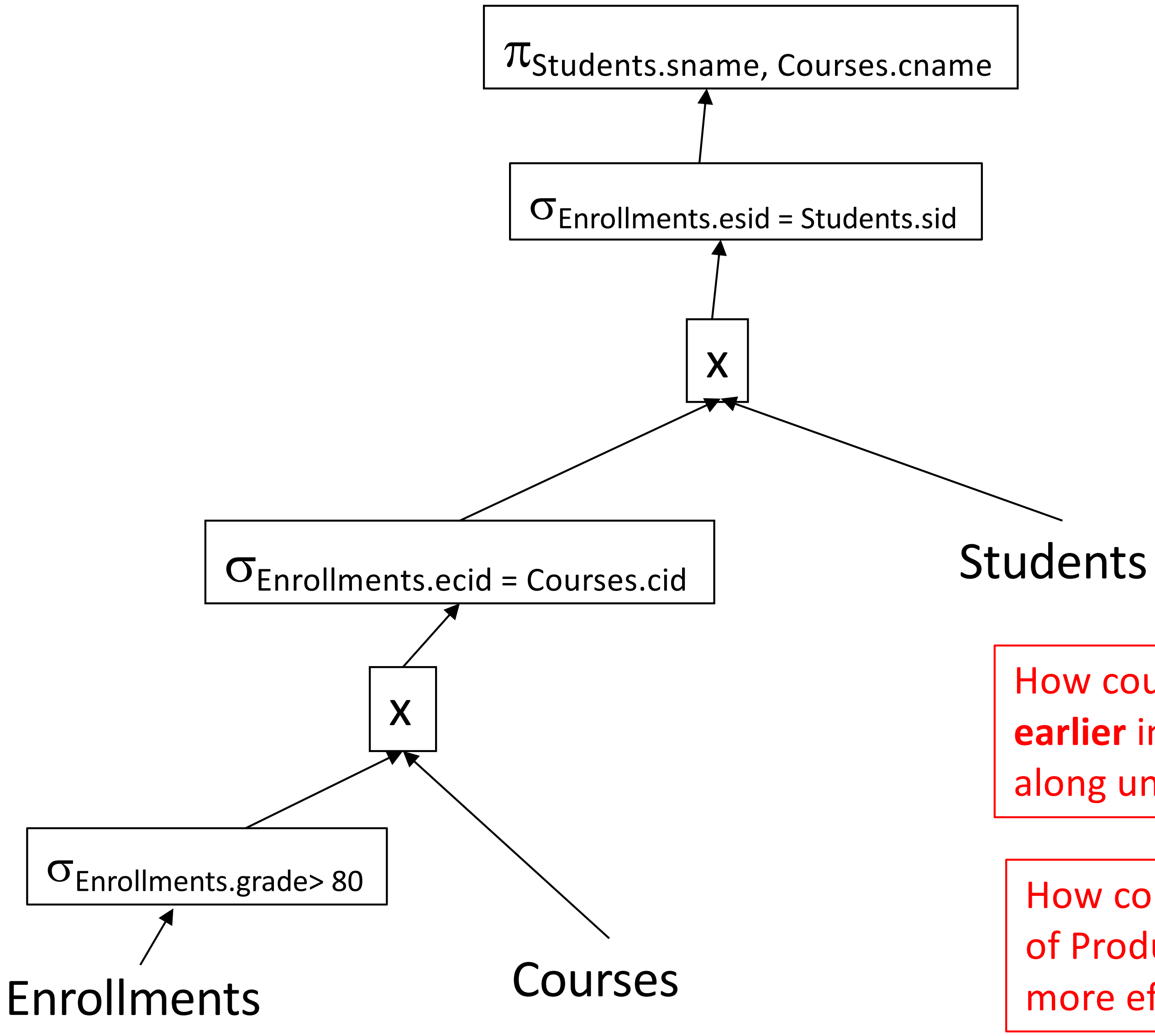


$\pi_{\text{Students.sname, Courses.cname}}$
$\sigma_{\text{Enrollments.esid = Students.sid}}$
x
$\sigma_{\text{Enrollments.ecid = Courses.cid AND Enrollments.grade>80}}$
Students
x
Enrollments
Courses

$\sigma_{C1 \text{ AND } C2 \text{ AND } C3}$ (E x C x S) =
σ_{C1} ($\sigma_{C2 \text{ AND } C3}$ (E x C x S)) =
σ_{C1} ($\sigma_{C2 \text{ AND } C3}$ (E x C) x S)

A <u>Third</u> Execution Plan for Query 2

`



$\pi_{\text{Students.sname, Courses.cname}}$
$\sigma_{\text{Enrollments.esid = Students.sid}}$
x
$\sigma_{\text{Enrollments.ecid = Courses.cid}}$
Students
x
$\sigma_{\text{Enrollments.grade> 80}}$
Enrollments
Courses

How could we do projections **earlier** in plan to avoid carrying along unnecessary attributes?

How could we do **Joins**, instead of Products to make plan more efficient?

Query Transformations

- What were some of the query equivalences that we talked about earlier?

- What other query equivalences do you know about?

Execution Plans

- When do you do SELECTION?
  - Predicate pushdown is always a good idea.
- How do you access each table?
  - Scan, index (which index), hash, …
- What's the order in which you Join tables?
  - Join/Equi-join is common; <u>avoid</u> Cartesian product
  - But which table do you start with?
    - Predicates on indexed columns are often useful in picking first table, then next table, to join, …
- What join method do you use for each join?
  - Nested loop join, merge join, hash-join, …
- How much parallelism do you use?
  - How do you schedule tasks to hardware?
- Do you need to sort? If so, when do you sort?

Query Optimization

- Comparing Execution Plans and finding a "good" (not necessarily best) plan
- Statistics that DBMS may keep to help calculate approximate query cost
  - Cardinality (number of rows) in table
  - Highest and lowest (non-null) value in column
  - Column cardinality (number of different values in column)
  - Number of appearances of the top 10 most frequent value in each column
  - Join cardinality between tables for particular equi-join
    - May be calculated, not stored; not well-defined if there are conditions (predicates) on the tables
  - Many other statistics are calculated approximately
- How frequently are stored statistics updated?
- Cost: CPU? I/O? Network? How do these get combined to compare plans?

EXPLAIN Statement

- Shows information about query plan
  - Each DBMS that has EXPLAIN has its own variation
  - Try it with PostgreSQL
- You may want to try to rewrite query yourself to find better execution plan if Query Optimizer isn’t smart enough to do so
- Should Optimizer take advice from users?

Practice Homework 5

Sailors(<u>sid</u>, sname, rating, age) // sailor id, sailor name, rating, age

Boats(<u>bid</u>, bname, color) // boat id, boat name, color of boat

Reserves(<u>sid, bid</u>, day) // sailor id, boat id, date that sid reserved bid.

- Use **Relational Algebra** to write the following **8 queries**.
- How might you optimize execution of queries using ideas in this Lecture, per discussion in slides 44-48?

1. Find the names of sailors who reserved boat 103.

2. Find the colors of boats reserved by Lubber.

3. Find the names of sailors who reserved at least one boat.

Practice Homework 5 (cont'd)

4. Find the names of sailors whose age > 20 and who have not reserved any boats.

5. Find the names of sailors who have reserved a red or a green boat.

6. Find the names of sailors who have reserved a red and a green boat.

7. Find the names of sailors who have reserved at least 2 different boats.

8. Find the names of sailors who have reserved exactly 2 different boats.