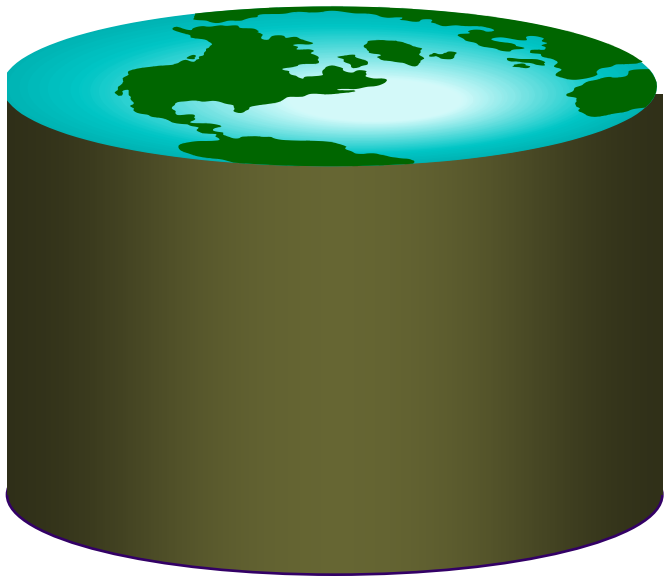
CS 405G: Introduction to Database Systems

Relational Algebra



Review

- Database
  - Relation schemas, relation instances and relational constraints.
- What's next?
  - Relational query language.
- Reading
  - Chapter 6.1~6.5

Relational Query Languages

- *Query languages*: Allow manipulation and retrieval of data from a database.
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic.
  - Allows for much optimization.
- Query Languages != programming languages!
  - QLs not intended to be used for complex calculations and inference (e.g. logical reasoning)
  - QLs support easy, efficient access to large data sets.

Formal Relational Query Languages

Two mathematical Query Languages form the basis for "real" languages (e.g. SQL), and for implementation:

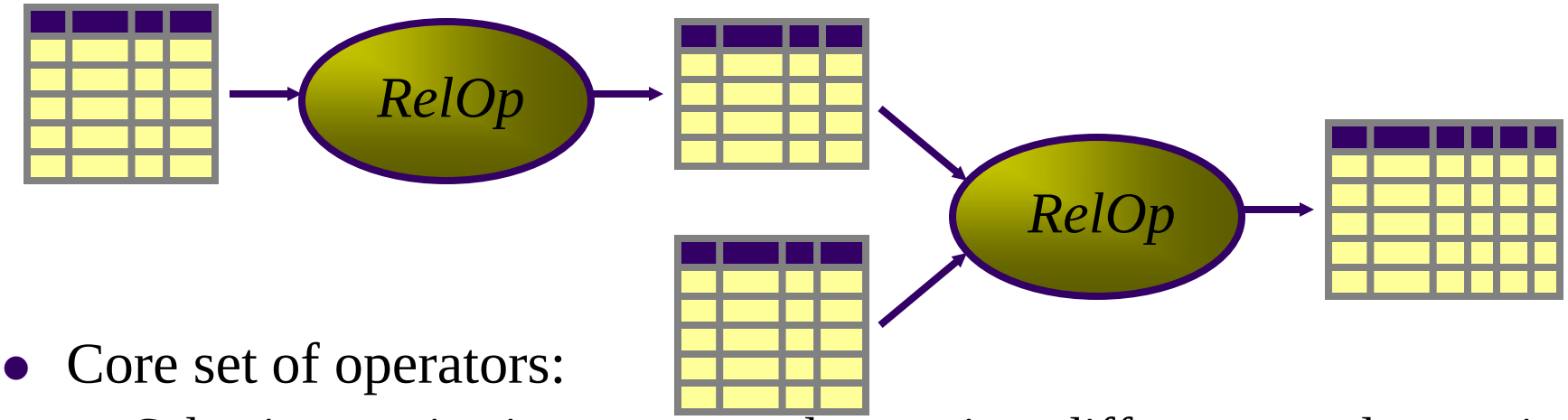
Relational Algebra: More operational, very useful for representing execution plans.

Relational Calculus: Lets users describe what they want, rather than how to compute it. (Non-procedural, *declarative*.)

✉ *Understanding Algebra & Calculus is key to understanding SQL, query processing!*

Relational algebra

A language for querying relational databases based on operators:



- Core set of operators:
  - Selection, projection, cross product, union, difference, and renaming
- Additional, derived operators:
  - Join, natural join, intersection, etc.
- Compose operators to make complex queries

Selection

- Input: a table R
- Notation: $\sigma_p R$
  - p is called a selection condition/predicate
- Purpose: filter rows according to some criteria
- Output: same columns as R, but only rows of R that satisfy p

Selection example

- Students with GPA higher than 3.0

$\sigma_{GPA > 3.0}\ Student$

sid	name	age	gpa
1234	John Smith	21	3.5
1123	Mary Carter	22	3.8
1011	Bob Lee	22	2.6
1204	Susan Wong	22	3.4
1306	Kevin Kim	21	2.9

sid	name	age	gpa
1234	John Smith	21	3.5
1123	Mary Carter	22	3.8
1204	Susan Wong	22	3.4

More on selection

- Selection predicate in general can include any column of *R*, constants, comparisons (=, ·, etc.), and Boolean connectives (∧: and, ∨: or, and ∷ not)
  - Example: straight A students under 18 or over 21

$$\sigma_{GPA = 4.0 \wedge (age < 18 \vee age > 21)} Student$$

- But you must be able to evaluate the predicate over a single row of the input table
  - Example: student with the highest GPA

~~σ_{GPA}, all *GPA* in *Student* table~~ *Student*

Projection

- Input: a table R
- Notation: $\pi_L R$
  - L is a list of columns in R
- Purpose: select columns to output
- Output: same rows, but only the columns in L
  - Order of the rows is preserved
  - Number of rows may be less (depends on where we have duplicates or not)

Projection example

- ID's and names of all students

$\pi_{SID,\ name}\ Student$

sid	name	age	gpa
1234	John Smith	21	3.5
1123	Mary Carter	22	3.8
1011	Bob Lee	22	2.6
1204	Susan Wong	22	3.4
1306	Kevin Kim	21	2.9

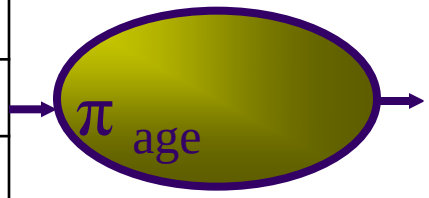
sid	name
1234	John Smith
1123	Mary Carter
1011	Bob Lee
1204	Susan Wong
1306	Kevin Kim

More on projection

- Duplicate output rows are removed (by definition)
  - Example: student ages

$\pi_{age}\ Student$

sid	name	age	gpa
1234	John Smith	21	3.5
1123	Mary Carter	22	3.8
1011	Bob Lee	22	2.6
1204	Susan Wong	22	3.4
1306	Kevin Kim	21	2.9



age
21
22

Cross product

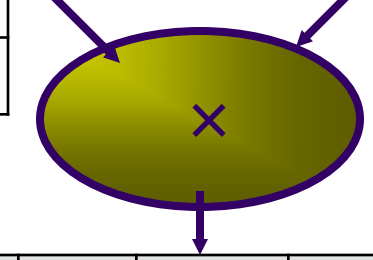
- Input: two tables *R* and *S*
- Notation: $R \times S$
- Purpose: pairs rows from two tables
- Output: for each row *r* in *R* and each row *s* in *S*, output a row *rs* (concatenation of *r* and *s*)

Cross product example

- *Student* X *Enroll*

sid	name	age	gpa
1234	John Smith	21	3.5
1123	Mary Carter	22	3.8
1011	Bob Lee	22	2.6

sid	cid	grade
1234	647	A
1123	108	A



sid	name	age	gpa	sid	cid	grade
1234	John Smith	21	3.5	1234	647	A
1123	Mary Carter	22	3.8	1234	647	A
1011	Bob Lee	22	2.6	1234	647	A
1234	John Smith	21	3.5	1123	108	A
1123	Mary Carter	22	3.8	1123	108	A
1011	Bob Lee	22	2.6	1123	108	A

A note on column ordering

- The ordering of columns in a table is considered unimportant (as is the ordering of rows)

sid	name	age	gpa
1234	John Smith	21	3.5
1123	Mary Carter	22	3.8
1011	Bob Lee	22	2.6

=

sid	name	gpa	age
1234	John Smith	3.5	21
1123	Mary Carter	3.8	22
1011	Bob Lee	2.6	22

- That means cross product is commutative, i.e., $R \times S = S \times R$ for any R and S

Derived operator: join

- Input: two tables R and S
- Notation: $R \bowtie_p S$
  - p is called a join condition/predicate
- Purpose: relate rows from two tables according to some criteria
- Output: for each row r in R and each row s in S, output a row rs if r and s satisfy p

- Shorthand for $\sigma_p (R \times S)$

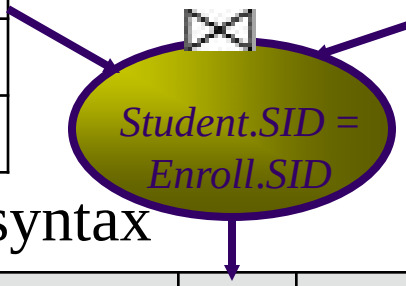
Join example

- Info about students, plus CID's of their courses

$$Student \bowtie_{Student.SID = Enroll.SID} Enroll$$

sid	name	age	gpa
1234	John Smith	21	3.5
1123	Mary Carter	22	3.8
1011	Bob Lee	22	2.6

sid	cid	grade
1234	647	A
1123	108	A



Use *table_name*. *column_name* syntax to disambiguate identically named columns from different input tables

sid	name	age	gpa	sid	cid	grade
1234	John Smith	21	3.5	1234	647	A
1123	Mary Carter	22	3.8	1123	108	A

Derived operator: natural join

- Input: two tables R and S
- Notation: $R * S$
- Purpose: relate rows from two tables, and
  - Enforce equality on all common attributes
  - Eliminate one copy of common attributes

- Shorthand for $\pi_L (R \bowtie_p S)$, where
  - p equates all attributes common to R and S
  - L is the union of all attributes from R and S, with duplicate attributes removed

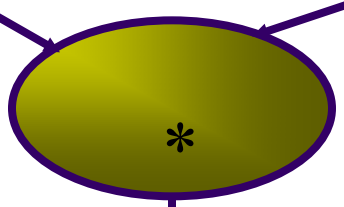
Natural join example

- $Student * Enroll = \pi_L (Student \bowtie_p Enroll)$

  $= \pi_{SID,\ name,\ age,\ GPA,\ CID} (Student \ !_{Student.SID\ =\ Enroll.SID} Enroll)$

sid	name	age	gpa
1234	John Smith	21	3.5
1123	Mary Carter	22	3.8
1011	Bob Lee	22	2.6

sid	cid	grade
1234	647	A
1123	108	A



sid	name	age	gpa	cid	grade
1234	John Smith	21	3.5	647	A
1123	Mary Carter	22	3.8	108	A

Union

- Input: two tables R and S
- Notation: $R \cup S$
  - R and S must have identical schema
- Output:
  - Has the same schema as R and S
  - Contains all rows in R and all rows in S, with duplicate rows eliminated

Difference

- Input: two tables *R* and *S*
- Notation: *R* - *S*
  - *R* and *S* must have identical schema
- Output:
  - Has the same schema as *R* and *S*
  - Contains all rows in *R* that are not found in *S*

Derived operator: intersection

- Input: two tables R and S
- Notation: $R \setminus S$
  - R and S must have identical schema
- Output:
  - Has the same schema as R and S
  - Contains all rows that are in both R and S
- Shorthand for $R - (R - S)$
- Also equivalent to $S - (S - R)$
- And to $R * S$

Renaming

- Input: a table R
- Notation: $\rho_S R$, $\rho_{(A_1, A_2, \ldots)} R$ or $\rho_{S(A_1, A_2, \ldots)} R$
- Purpose: rename a table and/or its columns
- Output: a renamed table with the same rows as R
- Used to
  - Avoid confusion caused by identical column names
  - Create identical columns names for natural joins

Renaming Example

- $\rho_{Enroll1}(SID1, CID1, Grade1)$ Enroll

sid	cid	grade
1234	647	A
1123	108	A

sid1	cid1	grade1
1234	647	A
1123	108	A

Review: Summary of core operators

- Selection: $\sigma_p R$
- Projection: $\pi_L R$
- Cross product: $R \times S$
- Union: $R \cup S$
- Difference: $R - S$
- Renaming: $\rho_{S(A_1, A_2, \ldots)} R$
  - Does not really add "processing" power

Review Summary of derived operators

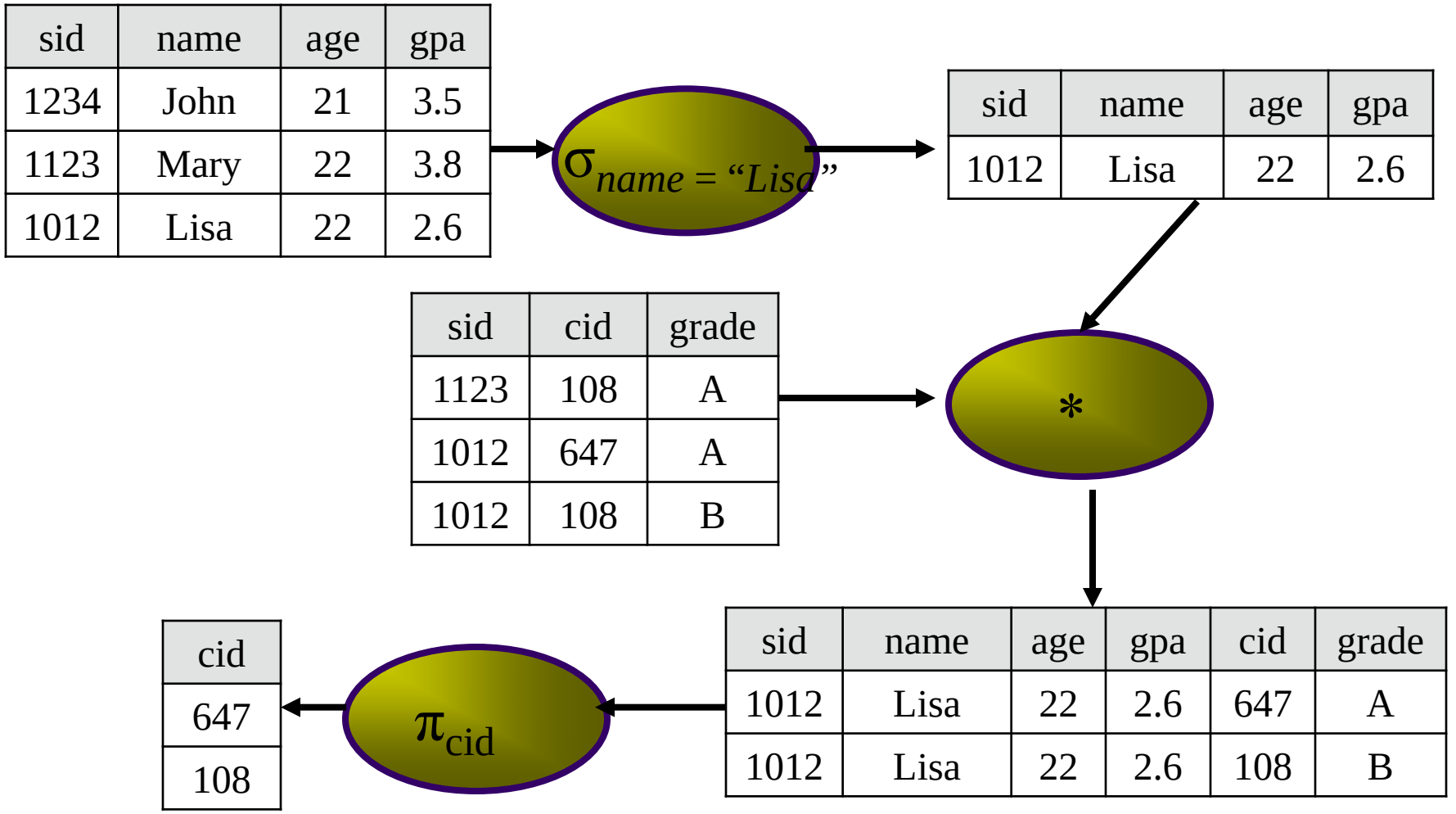
- Join: $R \bowtie_p S$
- Natural join: $R * S$
- Intersection: $R \cap S$

- Many more
  - Outer join, Division,
  - Semijoin, anti-semijoin, …

Using Join

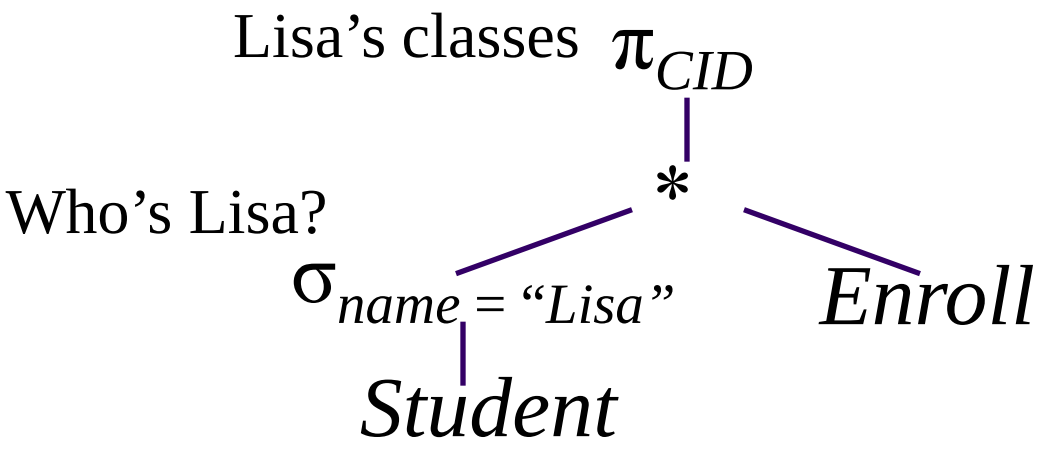
- Which classes is Lisa taking?
  - Student(sid: string, name: string, gpa: float)
  - Course(cid: string, department: string)
  - Enrolled(sid: string, cid: string, grade: character)
- An Answer:
  - Student_Lisa $\leftarrow \sigma_{name = \text{“}Lisa\text{”}} Student$
  - Lisa_Enrolled $\leftarrow$ Student_Lisa $*$Enrolled
  - Lisa's classes $\leftarrow \pi_{CID}$ Lisa_Enrolled
- Or:
  - Student_Enrolled $\leftarrow$ Student $*$Enrolled
  - Lisa_Enrolled $\leftarrow \sigma_{name = \text{“}Lisa\text{”}}$ Student_Enrolled
  - Lisa's classes $\leftarrow \pi_{CID}$ Lisa_Enrolled

Join Example



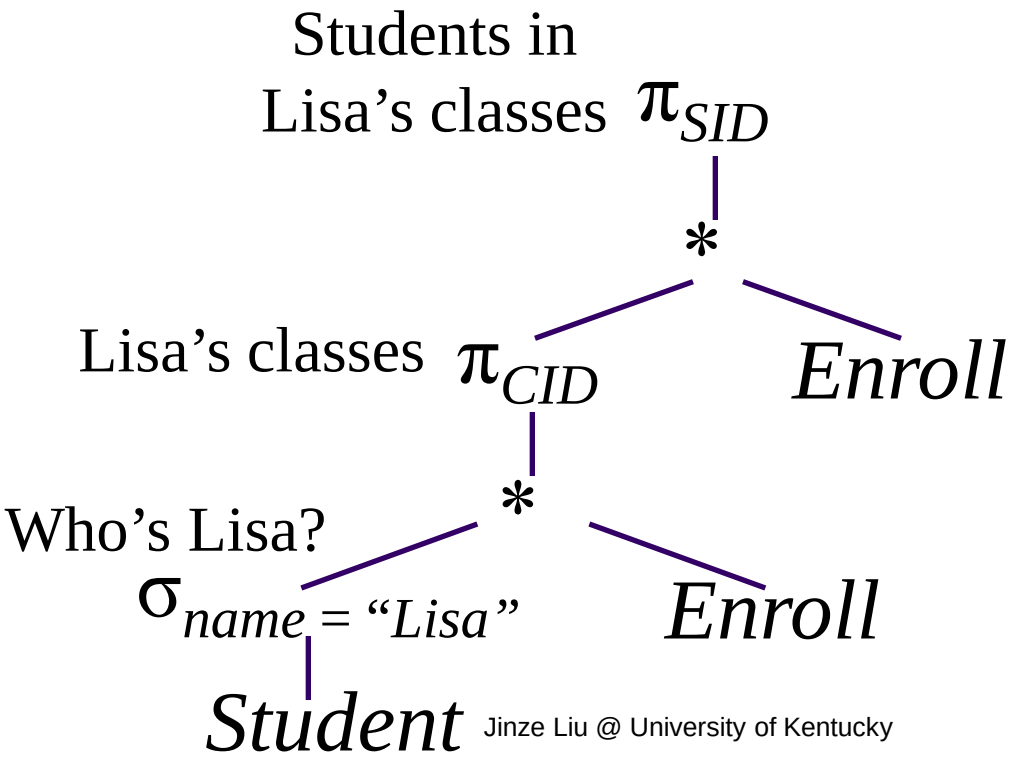
Lisa's Class

- $\pi_{CID}((\sigma_{name = \text{“}Lisa\text{”}} Student) * \text{Enrolled})$



Students in Lisa's Classes

- SID of Students in Lisa's classes
  - Student_Lisa $\leftarrow \sigma_{name = \text{“}Lisa\text{”}} Student$
  - Lisa_Enrolled $\leftarrow$ Student_Lisa $*$Enrolled
  - Lisa's classes $\leftarrow \pi_{CID}$ Lisa_Enrolled
  - Enrollment in Lisa's classes $\leftarrow$ Lisa's classes $*$Enrolled
  - Students in Lisa's class $\leftarrow \pi_{SID}$ Enrollment in Lisa's classes

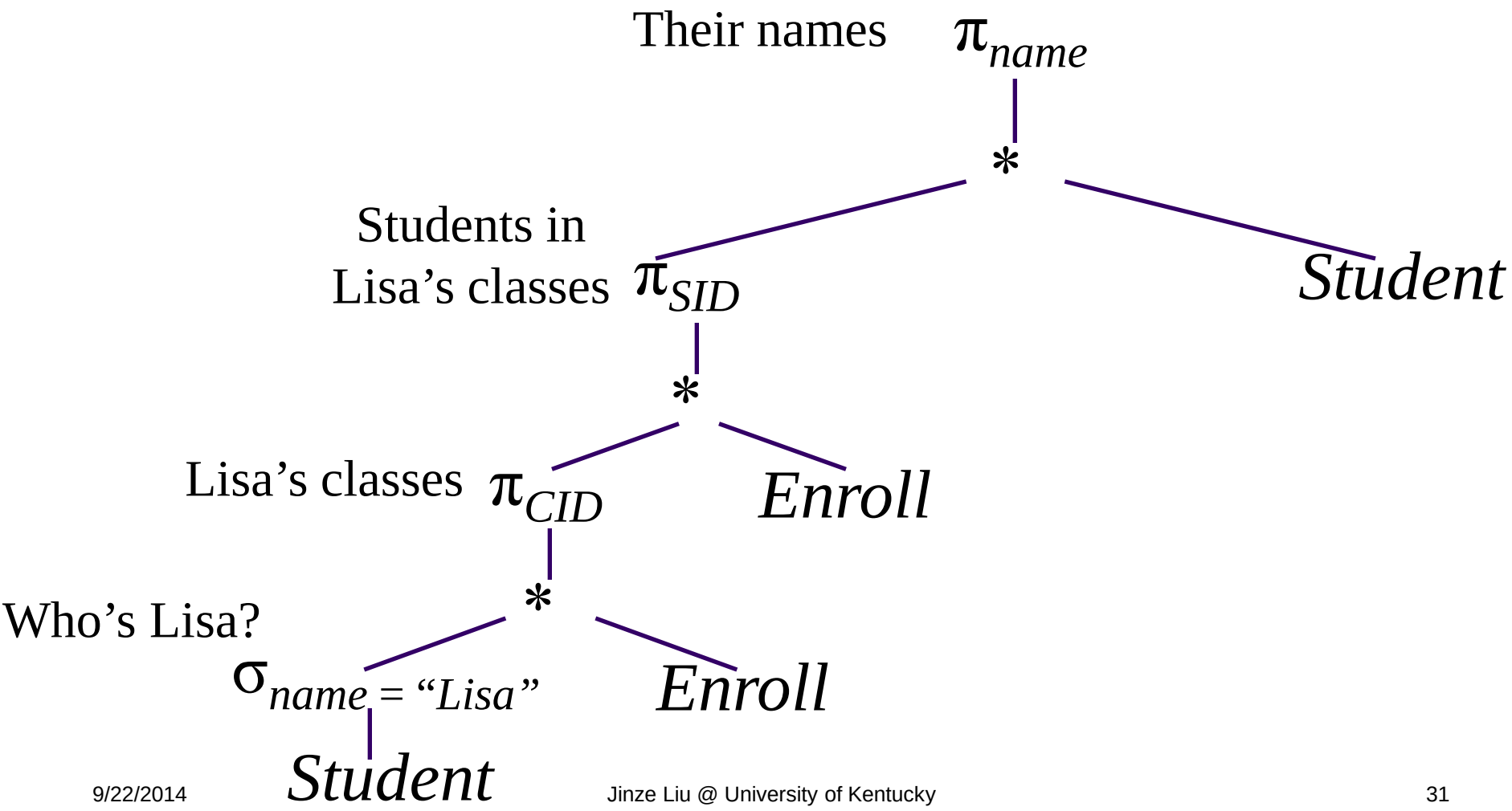


Tips in Relational Algebra

- Use temporary variables
- Use foreign keys to join tables

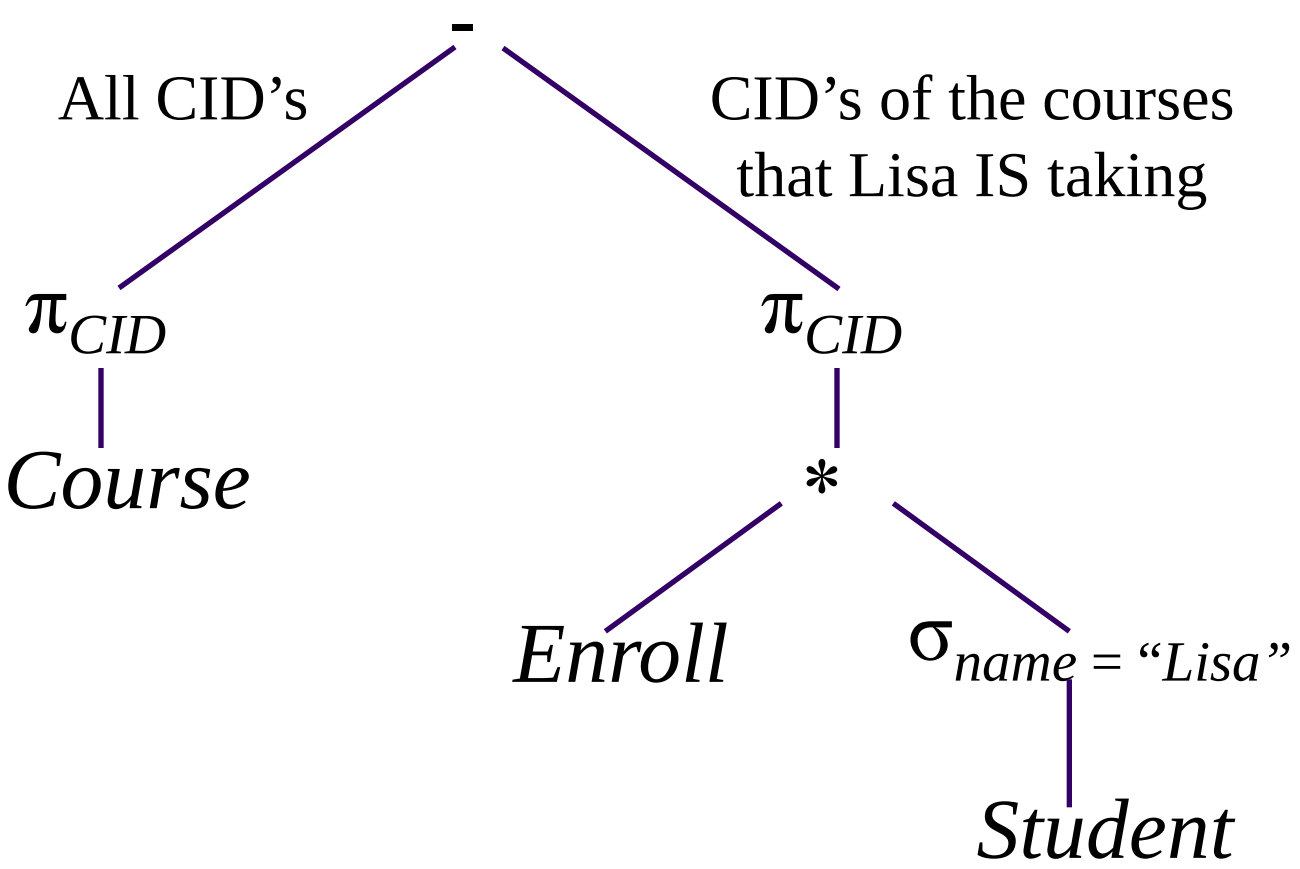
An exercise

- Names of students in Lisa's classes



Set Minus Operation

- CID's of the courses that Lisa is NOT taking



Renaming Operation

- $\rho_{Enrolled1}(SID1, CID1, Grade1)$ Enrolled

sid	cid	grade
1234	647	A
1123	108	A

sid1	cid1	grade1
1234	647	A
1123	108	A

Example

- We have the following relational schemas
  - Student(sid: string, name: string, gpa: float)
  - Course(cid: string, department: string)
  - Enrolled(sid: string, cid: string, grade: character)
- SID's of students who take at least two courses

$Enrolled \bowtie Enrolled$

$\pi_{SID}\ (Enrolled \bowtie_{Enrolled.SID = Enrolled.SID\ \&\ Enrolled.CID \neq Enrolled.CID} Enrolled)$

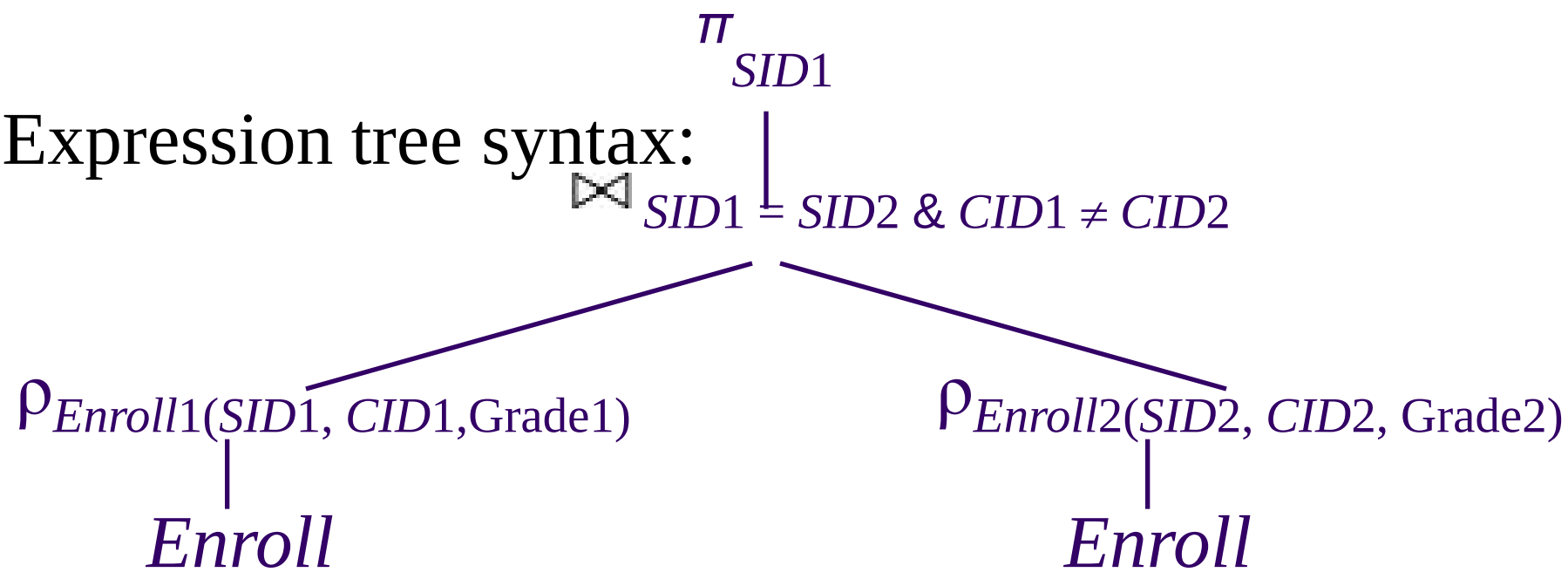
Example (cont.)

$\rho_{Enroll1}(SID1, CID1, Grade1)$ Enrolled

$\rho_{Enroll2}(SID2, CID2, Grade2)$ Enrolled

π_{SID} ($Enroll1 \bowtie_{SID1 = SID2 \,\&\, CID1 \neq CID2} Enroll2$)

Expression tree syntax:



How does it work?

sid	cid	grade
1123	108	A
1012	647	A
1012	108	B

sid2	cid2	grade2
1123	108	A
1012	647	A
1012	108	B

$Enroll1 \bowtie_{SID1 = SID2} Enroll2$

sid	cid	grade	sid2	cid2	grade2
1123	108	A	1123	108	A
1012	647	A	1012	647	A
1012	647	A	1012	108	B
1012	108	B	1012	647	A
1012	108	B	1012	108	B

sid	cid	grade
1123	108	A
1012	647	A
1012	108	B

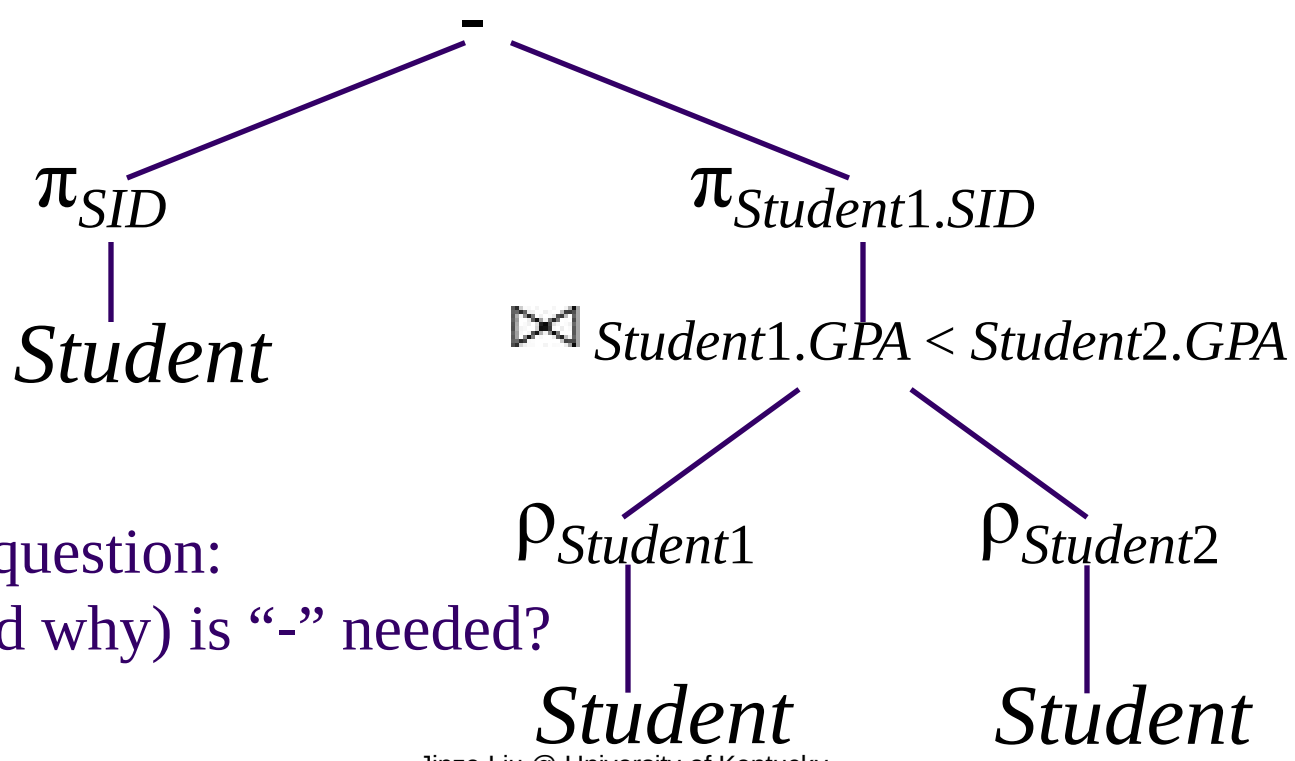
sid2	cid2	grade2
1123	108	A
1012	647	A
1012	108	B

$Enroll1 \bowtie_{SID1 = SID2 \ \& \ CID1 \neq CID2} Enroll2$

sid	cid	grade	sid2	cid2	grade2
~~1123~~	~~108~~	~~A~~	~~1123~~	~~108~~	~~A~~
~~1012~~	~~647~~	~~A~~	~~1012~~	~~647~~	~~A~~
1012	647	A	1012	108	B
1012	108	B	1012	647	A
~~1012~~	~~108~~	~~B~~	~~1012~~	~~108~~	~~B~~

A trickier exercise

- Who has the highest GPA?
  - Who has a GPA?
  - Who does NOT have the highest GPA?
    - Whose GPA is lower than somebody else's?



A deeper question:
When (and why) is "-" needed?

Tips in Relational Algebra

- A comparison is to identify a relationship

Review: Summary of core operators

- Selection: $\sigma_p R$
- Projection: $\pi_L R$
- Cross product: $R \times S$
- Union: $R \cup S$
- Difference: $R - S$
- Renaming: $\rho_{S(A_1, A_2, \ldots)} R$
  - Does not really add "processing" power

Review: Summary of derived operators

- Join: $R \bowtie_p S$
- Natural join: $R * S$
- Intersection: $R \cap S$

Review

- Relational algebra
  - Use temporary variable
  - Use foreign key to join relations
  - A comparison is to identify a relationship

Exercises of R. A.

Reserves

sid	bid	day
22	101	10/10/96
58	103	11/12/96

Boats

bid	bname	color
101	Interlake	Blue
102	Interlake	Red
103	Clipper	Green
104	Marine	Red

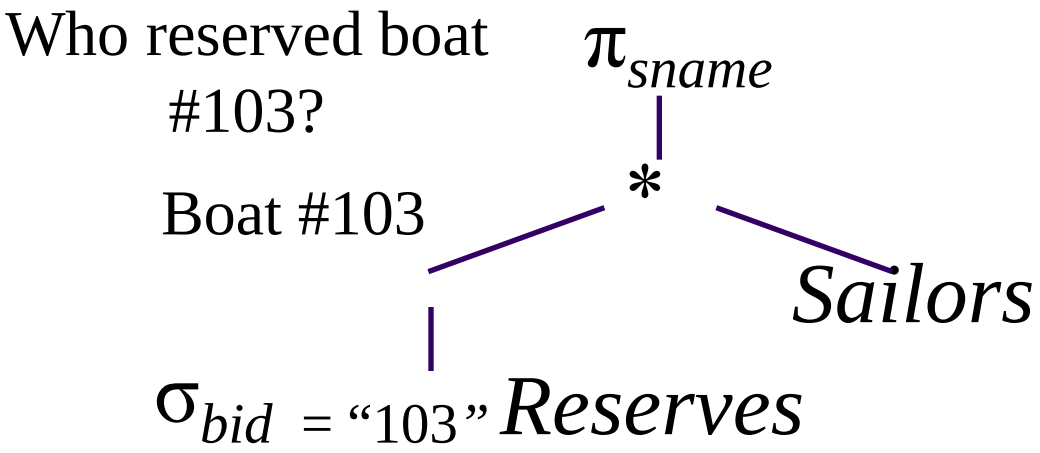
Sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Problem 1 Find names of sailors who've reserved boat #103

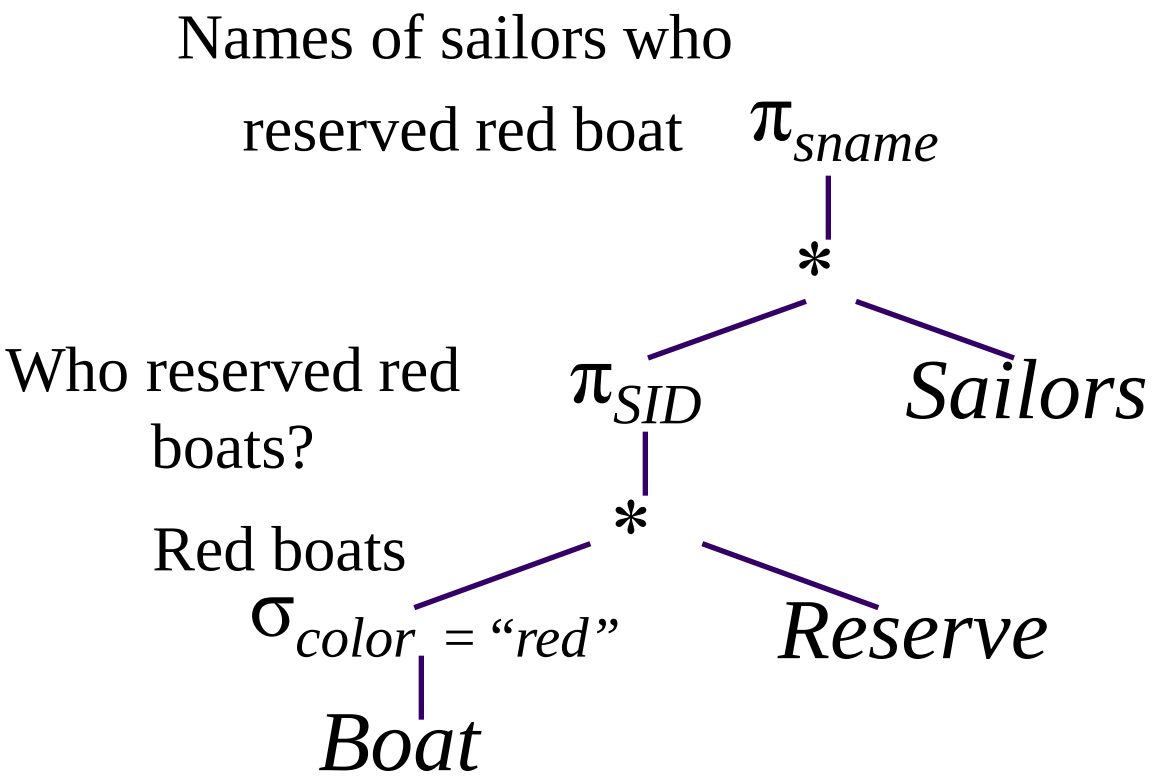
- Solution:

$$\pi_{sname}((\sigma_{bid=103}\text{Re}serves)*Sailors)$$



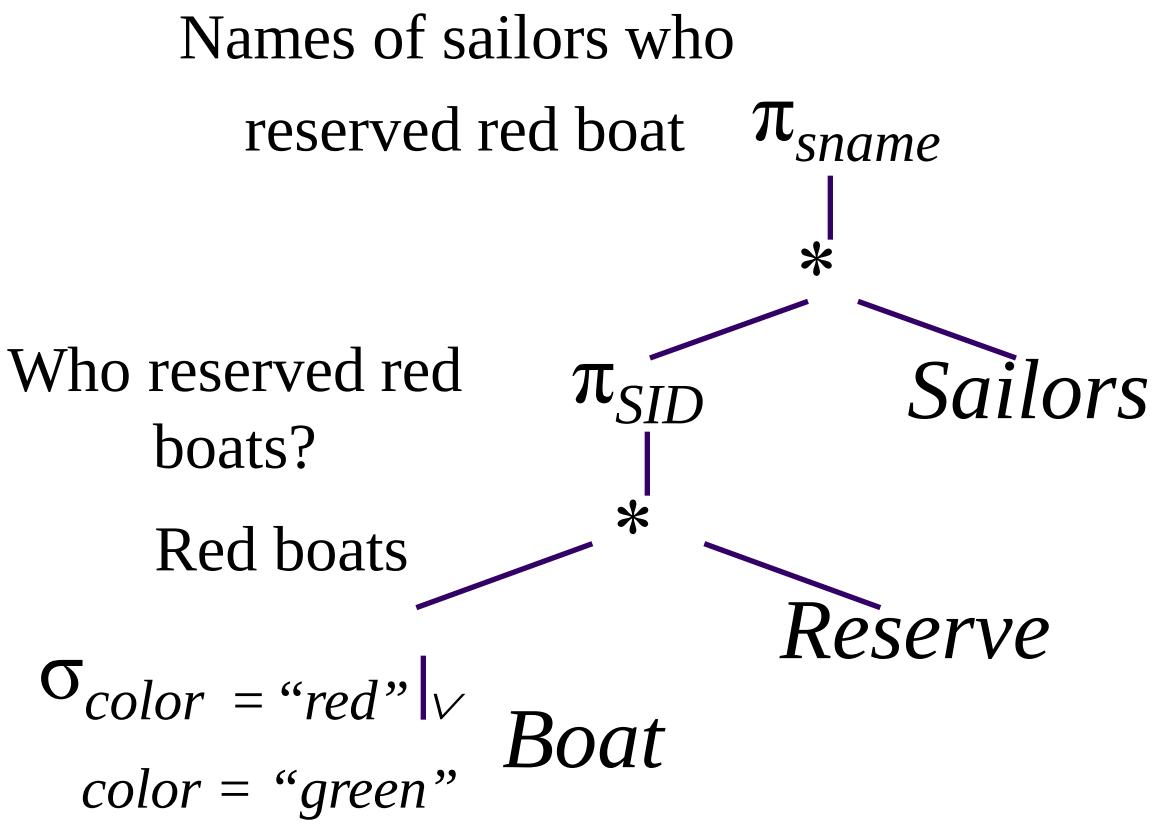
Problem 2: Find names of sailors who've reserved a red boat

- Information about boat color only available in Boats; so need an extra join:



Problem 3: Find names of sailors who've reserved a red boat or a green boat

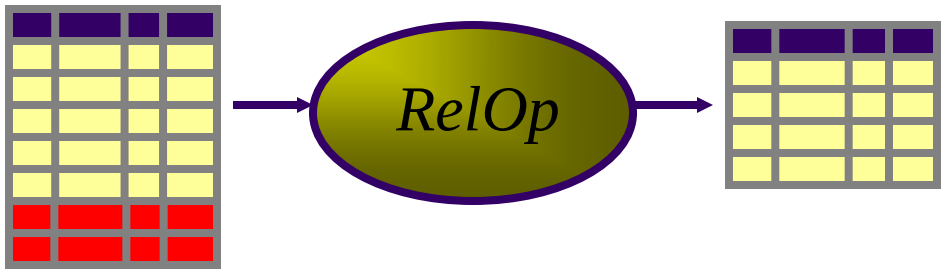
- Can identify all red or green boats, then find sailors who've reserved one of these boats:



Problem 4: Find names of sailors who've reserved only one boat

Monotone operators

Add more rows to the input...



What happens to the output?

- If some old output rows may need to be removed
  - Then the operator is non-monotone
- Otherwise the operator is monotone
  - That is, old output rows always remain "correct" when more rows are added to the input
- Formally, for a monotone operator *op*:
  R μ *R'* implies *op*(*R*) μ *op*(*R'*)

Why is "-" needed for highest GPA?

- Composition of monotone operators produces a monotone query
  - Old output rows remain "correct" when more rows are added to the input
- Highest-GPA query is non-monotone
  - Current highest GPA is 4.1
  - Add another GPA 4.2
  - Old answer is invalidated

☞ So it must use difference!

Classification of relational operators

- Selection: $\sigma_p R$ — Monotone
- Projection: $\pi_L R$ — Monotone
- Cross product: $R \times S$ — Monotone
- Join: $R \bowtie_p S$ — Monotone
- Natural join: $R * S$ — Monotone
- Union: $R \cup S$ — Monotone
- Difference: $R - S$ — Monotone w.r.t. R; non-monotone w.r.t S
- Intersection: $R \cap S$ — Monotone

Why do we need core operator *X*?

- Cross product
  - The only operator that adds columns
- Difference
  - The only non-monotone operator
- Union
  - The only operator that allows you to add rows?
- Selection? Projection?

Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- **Aggregate operation** in relational algebra

$$_{G1, G2, \ldots, Gn}\mathcal{g}_{F1(A1), F2(A2), \ldots, Fn(An)}(E)$$

  - E is any relational-algebra expression
  - $G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can be empty)
  - Each F_i is an aggregate function
  - Each A_i is an attribute name

Aggregate Operation – Example

- Relation r:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

$\mathcal{G}_{\textbf{sum(c)}}(r)$

sum-C
27

Aggregate Operation – Example

- Relation *account* grouped by *branch-name*:

branch-name	*account-number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$$branch\text{-}name\ \mathcal{G}\ sum(balance)\ (account)$$

branch-name	*balance*
Perryridge	1300
Brighton	1500
Redwood	700

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null*.
- Aggregate functions simply ignore null values
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same

Null Values

- Comparisons with null values return the special truth value *unknown*
  - If *false* was used instead of *unknown*, then *not (A < 5)* would not be equivalent to *A >= 5*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown)* = *unknown*
  - AND: *(true* **and** *unknown)* = *unknown*,
    (false **and** *unknown)* = *false*,
    (unknown **and** *unknown)* = *unknown*
  - NOT: *(***not** *unknown)* = *unknown*
- Result of select predicate is treated as *false* if it evaluates to *unknown*

Additional Operators

- Outer join
- Division

(Left) Outer Join

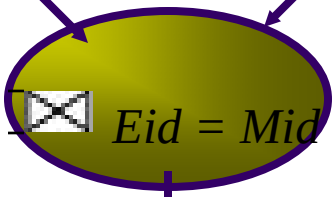
- Input: two tables R and S
- Notation: $R \ ⟕_p \ S$
- Purpose: pairs rows from two tables
- Output: for each row r in R and each row s in S,
  - if p satisfies, output a row rs (concatenation of r and s)
  - Otherwise, output a row r with NULLs
- Right outer join and full outer join are defined similarly

Left Outer Join Example

- *Employee* $\overline{\bowtie}$ *Eid = Mid* *Department*

Eid	Name
1234	John Smith
1123	Mary Carter
1011	Bob Lee

Did	Mid	Dname
4	1234	Research
5	1123	Finance



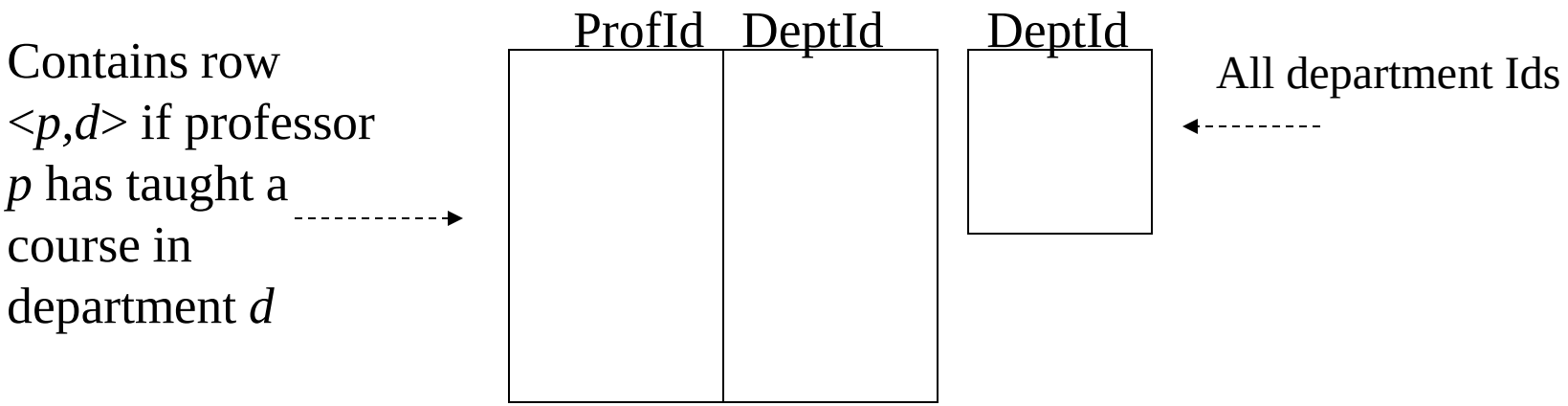
Eid	Name	Did	Mid	Dname
1234	John Smith	4	1234	Research
1123	Mary Carter	5	1123	Finance
1011	Bob Lee	NULL	NULL	NULL

Division Operator

- Input: two tables *R* and *S*
- Notation: $R \div S$
- Purpose: Find the subset of items in one set R that are related to *all* items in another set

Division Operator

- Find professors who have taught courses in *all* departments
  - Why does this involve division?



Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- All these operations are expressed using the assignment operator.

Deletion

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.
- Can delete only whole tuples; cannot delete values on only particular attributes
- A deletion is expressed in relational algebra by:

$$r \leftarrow r - E$$

where r is a relation and E is a relational algebra query.

Deletion Examples

- Delete all account records in the Perryridge branch.

  $account \leftarrow account - \sigma_{branch\_name = \text{“Perryridge”}}(account)$

- Delete all loan records with amount in the range of 0 to 50

  $loan \leftarrow loan - \sigma_{amount \geq 0 \text{ and } amount \leq 50}(loan)$

- Delete all accounts at branches located in Needham.

  $r_1 \leftarrow \sigma_{branch\_city = \text{“Needham”}}(account \bowtie branch)$

  $r_2 \leftarrow \Pi_{branch\_name,\ account\_number,\ balance}(r_1)$

  $r_3 \leftarrow \Pi_{customer\_name,\ account\_number}(r_2 \bowtie depositor)$

  $account \leftarrow account - r_2$

  $depositor \leftarrow depositor - r_3$

Insertion

- To insert data into a relation, we either:
  - specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted
- in relational algebra, an insertion is expressed by:

$$r \leftarrow r \cup E$$

  where r is a relation and E is a relational algebra expression.
- The insertion of a single tuple is expressed by letting E be a constant relation containing one tuple.

Insertion Examples

- Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.

  $account \leftarrow account \cup$ {("Perryridge", A-973, 1200)}

  depositor $\leftarrow depositor \cup$ {("Smith", A-973)}

- Provide as a gift for all loan customers in the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account.

  $r_1 \leftarrow (\sigma_{branch\_name = \text{"Perryridge"}} (borrower \bowtie loan))$

  $account \leftarrow account \cup \prod_{branch\_name,\ loan\_number,200} (r_1)$

  $\text{depositor} \leftarrow depositor \cup \prod_{customer\_name,\ loan\_number} (r_1)$

Updating

- A mechanism to change a value in a tuple without changing *all* values in the tuple
- Use the generalized projection operator to do this task

$$r \leftarrow \prod_{F_1, F_2, \ldots, F_l}(r)$$

- Each F_i is either
  - the i^{th} attribute of r, if the i^{th} attribute is not updated, or,
  - if the attribute is to be updated F_i is an expression, involving only constants and the attributes of r, which gives the new value for the attribute

Update Examples

- Make interest payments by increasing all balances by 5 percent.

$$account \leftarrow \Pi_{account\_number,\ branch\_name,\ balance\ *\ 1.05}(account)$$

- Pay all accounts with balances over $10,000 6 percent interest and pay all others 5 percent

$$account \leftarrow \Pi_{account\_number,\ branch\_name,\ balance\ *\ 1.06}(\sigma_{BAL > 10000}(account)) \cup \Pi_{account\_number,\ branch\_name,\ balance\ *\ 1.05}(\sigma_{BAL \leq 10000}(account))$$

What is “algebra”

- Mathematical model consisting of:
  - *Operands* --- Variables or values;
  - *Operators* --- Symbols denoting procedures that construct new values from a given values

- **Relational Algebra** is algebra whose operands are relations and operators are designed to do the most commons things that we need to do with relations

Why is r.a. a good query language?

- Simple
  - A small set of core operators who semantics are easy to grasp
- Declarative?
  - Yes, compared with older languages like CODASYL
  - Though operators do look somewhat “procedural”
- Complete?
  - With respect to what?

Review

- Expression tree
- Tips in writing R.A.
  - Use temporary variables
  - Use foreign keys to join tables
  - A comparison is to identify a relationship
  - Use set minus in non-monotonic results