Relational Model & Algebra

Peter Scheuermann

Example

Student

SID	*name*	*age*	*GPA*
142	Bart	19	2.3
123	Milhouse	21	3.1
857	Lisa	8	4.3
456	Ralph	8	2.3
…	…	…	…

Course

CID	*title*
CPS116	Intro. to Database Systems
CPS130	Analysis of Algorithms
CPS114	Computer Networks
…	…

Enroll

SID	*CID*
142	CPS116
142	CPS114
123	CPS116
857	CPS116
857	CPS130
456	CPS114
…	…

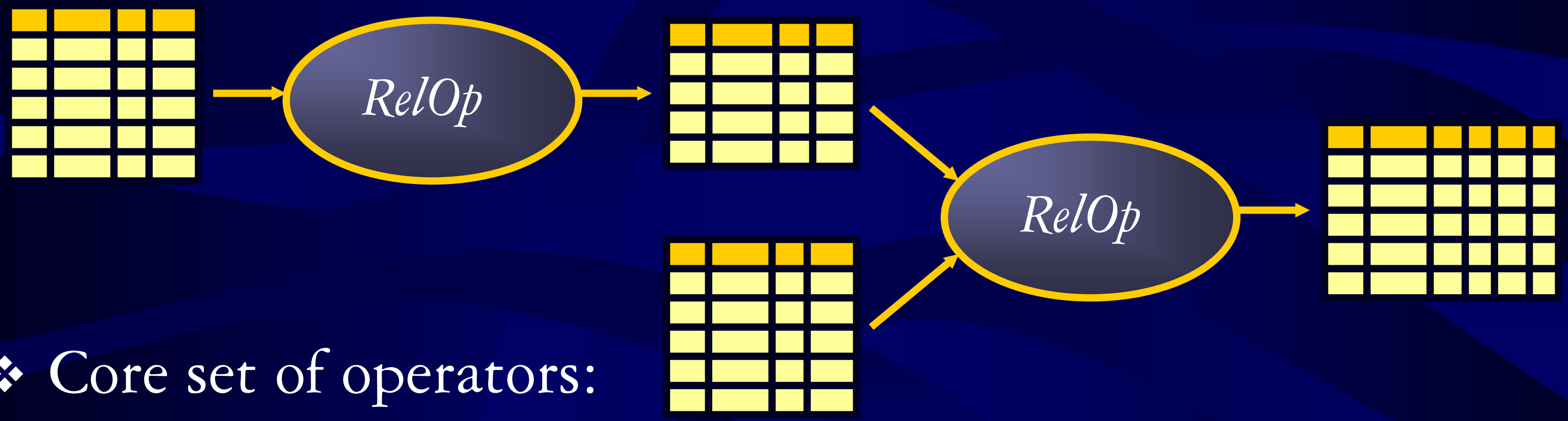
Ordering of rows doesn't matter (even though the output is always in *some* order)

Example

- Schema
  - *Student* (*SID* integer, *name* string, *age* integer, *GPA* float)
  - *Course* (*CID* string, *title* string)
  - *Enroll* (*SID* integer, *CID* integer)
- Instance
  - { ⟨142, Bart, 19, 2.3⟩, ⟨123, Milhouse, 21, 3.1⟩, ...}
  - { ⟨CPS116, Intro. to Database Systems⟩, ...}
  - { ⟨142, CPS116⟩, ⟨142, CPS114⟩, ...}

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

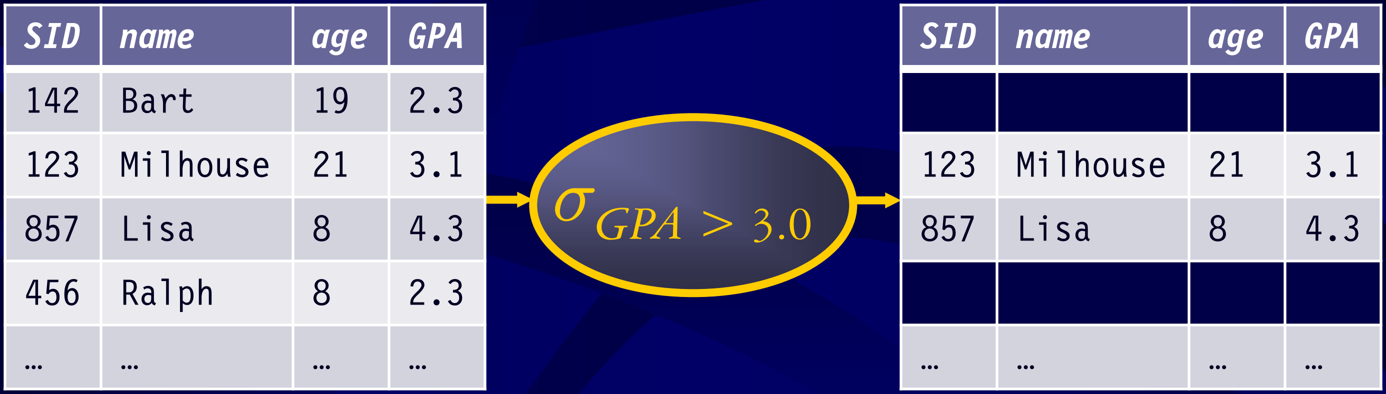
Selection

- 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

$\sigma_{GPA\,>\,3.0}\ \mathit{Student}$



SID	*name*	*age*	*GPA*
142	Bart	19	2.3
123	Milhouse	21	3.1
857	Lisa	8	4.3
456	Ralph	8	2.3
…	…	…	…

$\xrightarrow{\sigma_{GPA\,>\,3.0}}$

SID	*name*	*age*	*GPA*
123	Milhouse	21	3.1
857	Lisa	8	4.3
…	…	…	…

More on selection

- Selection predicate in general can include any column of R, constants, comparisons ($=$, $\leq$, etc.), and Boolean connectives ($\wedge$: and, $\vee$: or, and $\neg$: not)
  - Example: straight A students under 18 or over 21

    $\sigma_{GPA \geq 4.0 \wedge (age < 18 \vee age > 21)} \textit{Student}$
- But you must be able to evaluate the predicate over a single row of the input table
  - Example: student with the highest GPA

    $\sigma_{\xcancel{GPA \geq \text{all } GPA \text{ in } \textit{Student} \text{ table}}} \textit{Student}$

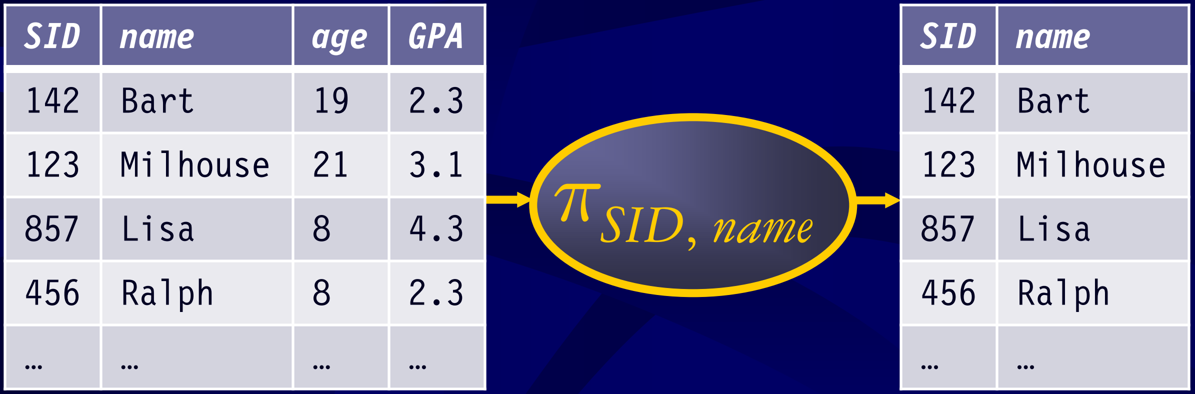
Projection

- 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

Projection example

❖ ID's and names of all students

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



SID	name	age	GPA
142	Bart	19	2.3
123	Milhouse	21	3.1
857	Lisa	8	4.3
456	Ralph	8	2.3
…	…	…	…

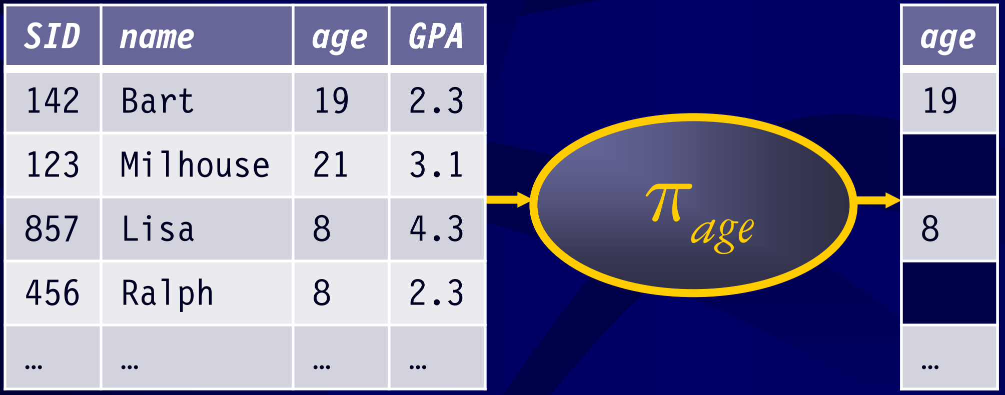
$\pi_{SID,\ name}$

SID	name
142	Bart
123	Milhouse
857	Lisa
456	Ralph
…	…

More on projection

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

$\pi_{age}\ Student$



SID	*name*	*age*	*GPA*
142	Bart	19	2.3
123	Milhouse	21	3.1
857	Lisa	8	4.3
456	Ralph	8	2.3
…	…	…	…

$$\xrightarrow{}\ \pi_{age}\ \xrightarrow{}$$

age
19
8
…

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* × *Enroll*

SID	name	age	GPA
142	Bart	19	2.3
123	Milhouse	21	3.1
…	…	…	…

SID	CID
142	CPS116
142	CPS114
123	CPS116
…	…

×

SID	name	age	GPA	SID	CID
142	Bart	19	2.3	142	CPS116
142	Bart	19	2.3	142	CPS114
142	Bart	19	2.3	123	CPS116
123	Milhouse	21	3.1	142	CPS116
123	Milhouse	21	3.1	142	CPS114
123	Milhouse	21	3.1	123	CPS116
…	…	…	…	…	…

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	SID	CID
142	Bart	10	2.3	142	CPS116
142	Bart	10	2.3	142	CPS114
142	Bart	10	2.3	123	CPS116
123	Milhouse	10	3.1	142	CPS116
123	Milhouse	10	3.1	142	CPS114
123	Milhouse	10	3.1	123	CPS116
…	…	…	…	…	…

$=$

SID	CID	SID	name	age	GPA
142	CPS116	142	Bart	10	2.3
142	CPS114	142	Bart	10	2.3
123	CPS116	142	Bart	10	2.3
142	CPS116	123	Milhouse	10	3.1
142	CPS114	123	Milhouse	10	3.1
123	CPS116	123	Milhouse	10	3.1
…	…	…	…	…	…

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

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: join

(A.k.a. "theta-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 $\sigma_p (R \times S)$

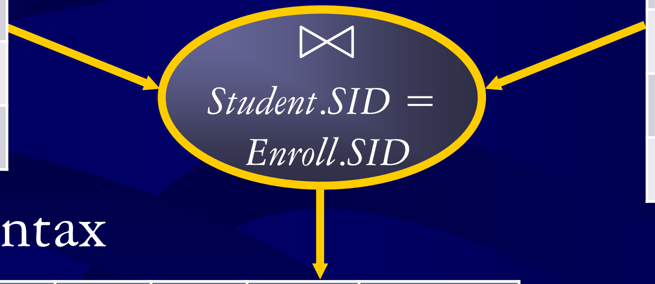
Join example

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

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

SID	name	age	GPA
142	Bart	19	2.3
123	Milhouse	21	3.1
…	…	…	…

SID	CID
142	CPS116
142	CPS114
123	CPS116
…	…



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

SID	name	age	GPA	SID	CID
142	Bart	19	2.3	142	CPS116
142	Bart	19	2.3	142	CPS114
123	Milhouse	21	3.1	123	CPS116
…	…	…	…	…	…

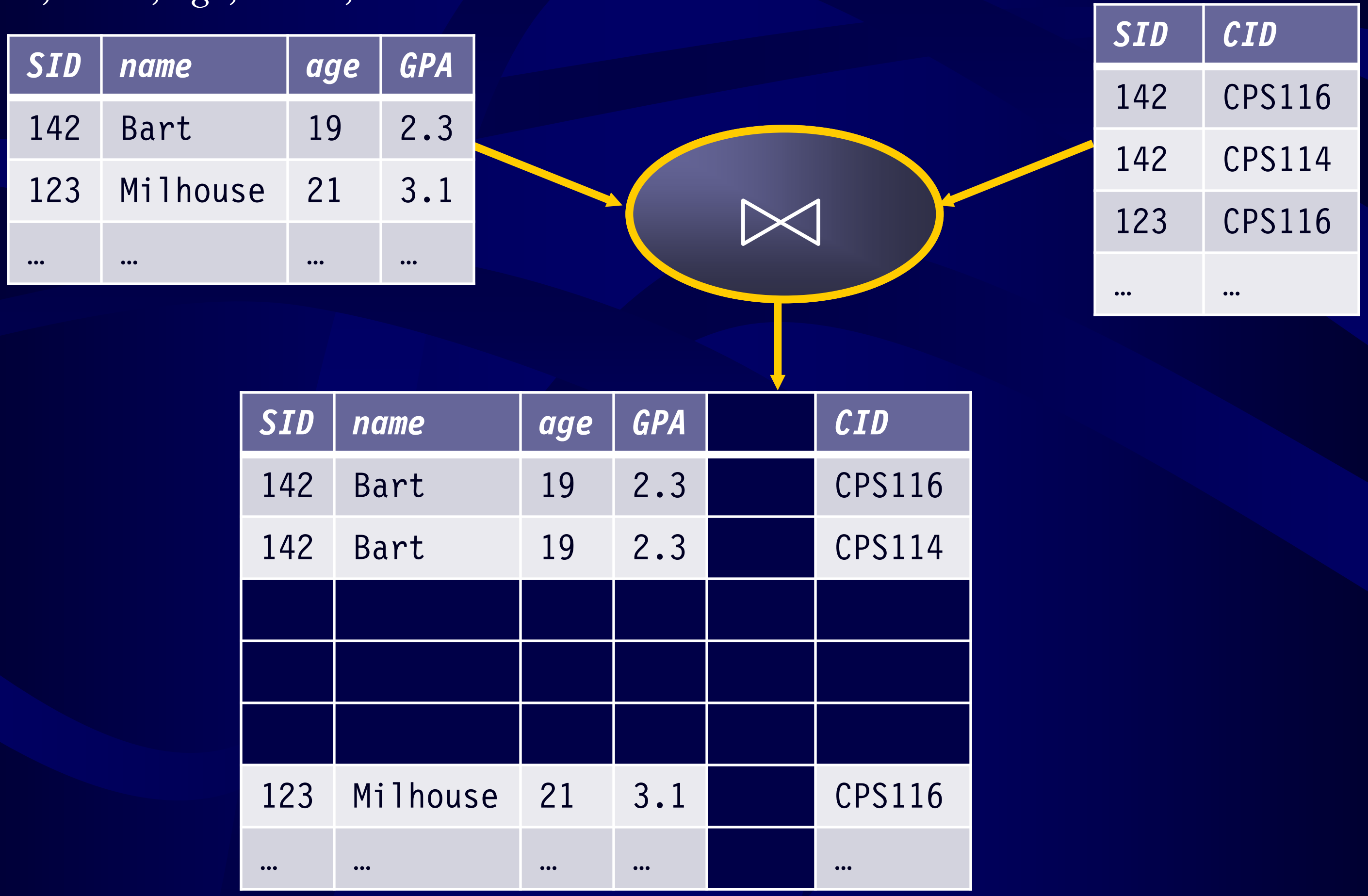
Derived operator: natural join

- Input: two tables *R* and *S*
- Notation: $R \bowtie S$
- 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

- $Student \bowtie Enroll = \pi_{?} (Student \bowtie_{?} Enroll)$

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



SID	name	age	GPA
142	Bart	19	2.3
123	Milhouse	21	3.1
…	…	…	…

SID	CID
142	CPS116
142	CPS114
123	CPS116
…	…

SID	name	age	GPA		CID
142	Bart	19	2.3		CPS116
142	Bart	19	2.3		CPS114
123	Milhouse	21	3.1		CPS116
…	…	…	…		…

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

❖ SID's of students who take at least two courses

$Enroll \bowtie_{?} Enroll$

$\pi_{SID}\,(Enroll \bowtie_{\cancel{Enroll.SID\, =\, Enroll.SID\, \wedge\, Enroll.CID\, \neq\, Enroll.CID}} Enroll)$

Expression tree syntax:

$$\begin{tikzcd}
 & \pi_{SID1} \arrow[d, dash] & \\
 & \bowtie_{SID1\, =\, SID2\, \wedge\, CID1\, \neq\, CID2} \arrow[dl, dash] \arrow[dr, dash] & \\
\rho_{Enroll1(SID1,\, CID1)} \arrow[d, dash] & & \rho_{Enroll2(SID2,\, CID2)} \arrow[d, dash] \\
Enroll & & Enroll
\end{tikzcd}$$

Summary of independent 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

Summary of derived operators

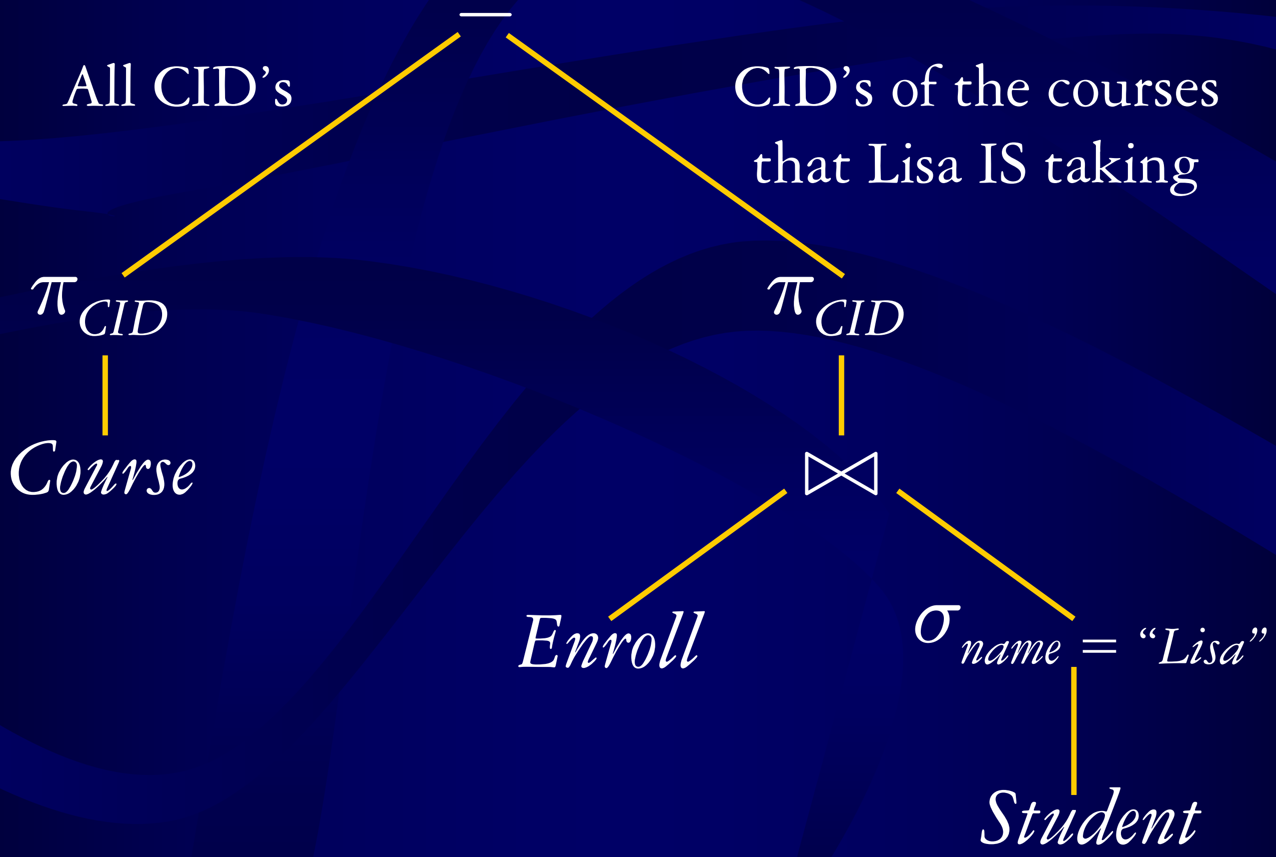
- Join: $R \bowtie_p S$
- Natural join: $R \bowtie S$
- Intersection: $R \cap S$

- Many more
  - Semijoin, outer join, inner join

Another exercise

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

Writing a query top-down:


−
All CID's
CID's of the courses that Lisa IS taking
π_{CID}
π_{CID}
Course
$\bowtie$
Enroll
$\sigma_{name = \text{“Lisa”}}$
Student

A trickier exercise

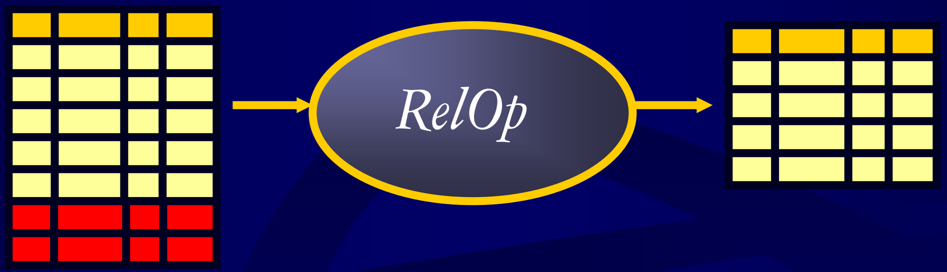
- Who has the highest GPA?
  - Who does NOT have the highest GPA?
  - Whose GPA is lower than somebody else's?

$$\begin{tikzcd}
 & & - \arrow[dll, dash] \arrow[dr, dash] & & \\
\pi_{SID} \arrow[d, dash] & & & \pi_{Student1.SID} \arrow[d, dash] & \\
Student & & & \bowtie_{Student1.GPA < Student2.GPA} \arrow[dl, dash] \arrow[dr, dash] & \\
 & & \rho_{Student1} \arrow[d, dash] & & \rho_{Student2} \arrow[d, dash] \\
 & & Student & & Student
\end{tikzcd}$$

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

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*:
  $R \subseteq R'$ implies $op(R) \subseteq op(R')$ for any R, R'

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: $R \bowtie S$	Monotone
❖ Union: $R \cup S$	Monotone
❖ Difference: $R - S$	Monotone or not ?
❖ Intersection: $R \cap S$	Monotone

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!

Division Operation

- Notation:
- Suited to queries that include the phrase "for all".
- Let r and s be relations on schemas R and S respectively where
  - $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S = (B_1, \ldots, B_n)$

The result of r ÷ s is a relation on schema

$R - S = (A_1, \ldots, A_m)$

$$r \div s = \{ t \mid t \in \Pi_{R-S}(r) \wedge \forall\, u \in s\, (tu \in r) \}$$

Where tu means the concatenation of tuples t and u to produce a single tuple

Division Operation – Example

- Relations *r*, *s*:

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
$\in$	6
$\in$	1
β	2

r

B
1
2

s

- *r* ÷ *s*:

A
α
β

Division Operation – Example

- Relations *r, s*:

⑩*A*	⑩*B*
⑩ α	⑩ *1*
⑩ α	⑩ *2*
⑩ α	⑩ *3*
⑩ β	⑩ *1*
⑩ γ	⑩ *1*
⑩ δ	⑩ *1*
⑩ δ	⑩ *3*
⑩ δ	⑩ *4*
⑩ $\in$	⑩ *6*
⑩ $\in$	⑩ *1*
⑩ β	⑩ *2*

⑩

⑩*B*
⑩ *1*
⑩ *2*

⑩*s*

- *r* ÷ *s*:

⑩ α
⑩ β

Extended Relational-Algebra-Operations

- Generalized Projection
- Aggregate Functions
- Outer Join

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that do not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist
  - All comparisons involving *null* are (roughly speaking) **false** by definition.
    - We shall study precise meaning of comparisons with nulls later

Outer Join – Example

- Relation *loan*

⑩*loan_number*	⑩*branch_name*	⑩*amount*
⑩L-170	⑩Downtown	⑩3000
⑩L-230	⑩Redwood	⑩4000
⑩L-260	⑩Perryridge	⑩1700

- Relation *borrower*

⑩*customer_name*	⑩*loan_number*
⑩Jones	⑩L-170
⑩Smith	⑩L-230
⑩Hayes	⑩L-155

Outer Join – Example

- Join

loan $\bowtie$ *borrower*

⑩*loan_number*	⑩*branch_name*	⑩*amount*	⑩*customer_name*
⑩L-170	⑩Downtown	⑩3000	⑩Jones
⑩L-230	⑩Redwood	⑩4000	⑩Smith

- Left Outer Join

loan ⟕ *borrower*

⑩*loan_number*	⑩*branch_name*	⑩*amount*	⑩*customer_name*
⑩L-170	⑩Downtown	⑩3000	⑩Jones
⑩L-230	⑩Redwood	⑩4000	⑩Smith
⑩L-260	⑩Perryridge	⑩1700	⑩*null*

Outer Join – Example

- Right Outer Join

  loan ⟖ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

- Full Outer Join

  loan ⟗ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

Relational calculus

- $\{\, s.SID \mid s \in Student \wedge \neg(\exists s' \in Student{:}\ s.GPA < s'.GPA) \,\}$, or
  $\{\, s.SID \mid s \in Student \wedge (\forall s' \in Student{:}\ s.GPA \geq s'.GPA) \,\}$
- Relational algebra = "safe" relational calculus
  - Every query expressible as a safe relational calculus query is also expressible as a relational algebra query
  - And vice versa
- Example of an unsafe relational calculus query
  - $\{\, s.name \mid \neg(s \in Student) \,\}$
  - Cannot evaluate this query just by looking at the database

Implementing Recursion

- Relational algebra has no recursion
  - Example of something not expressible in relational algebra: Given relation *Parent*(*parent*, *child*), who are Bart's ancestors?
- Why not include it ?
  - Optimization becomes undecidable
  - You can always implement it at the application level
- Recursion is added to SQL nevertheless!