Relational Model & Algebra

CPS 216

Advanced Database Systems

Announcements (January 18)

- ❖ Homework #1 will be assigned on Thursday
- ❖ Reading assignment for this week
  - Posted on course Web page
  - Review due on Thursday night

Relational data model

- ❖ A database is a collection of relations (or tables)
- ❖ Each relation has a list of attributes (or columns)
  - Set-valued attributes not allowed
- ❖ Each attribute has a domain (or type)
- ❖ Each relation contains a set of tuples (or rows)
  - Duplicates not allowed

☞ Simplicity is a virtue!

Example

Student

SID	*name*	*age*	*GPA*
142	Bart	10	2.3
123	Milhouse	10	3.1
857	Lisa	8	4.3
456	Ralph	8	2.3
...	...	...	...

Course

CID	*title*
CPS216	Advanced Database Systems
CPS230	Analysis of Algorithms
CPS214	Computer Networks
...	...

Enroll

SID	*CID*
142	CPS216
142	CPS214
123	CPS216
857	CPS216
857	CPS230
456	CPS214
...	...

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

Why did Codd call them "relations"?

Each *n*-tuple relates *n* elements from *n* domains, precisely in the mathematical sense of a "relation"

Schema versus instance

- ❖ Schema (metadata)
  - Specification of how data is to be structured logically
  - Defined at set-up
  - Rarely changes
- ❖ Instance
  - Content
  - Changes rapidly, but always conforms to the schema

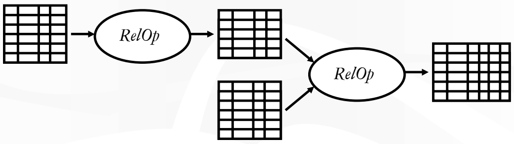
☞ Compare to type and object of type in a programming language

---

Example

- ❖ Schema
  - *Student* (*SID* integer, *name* string, *age* integer, *GPA* float)
  - *Course* (*CID* string, *title* string)
  - *Enroll* (*SID* integer, *CID* integer)
- ❖ Instance
  - { ⟨142, Bart, 10, 2.3⟩, ⟨123, Milhouse, 10, 3.1⟩, ...}
  - { ⟨CPS216, Advanced Database Systems⟩, ...}
  - { ⟨142, CPS216⟩, ⟨142, CPS214⟩, ...}

Relational algebra operators



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

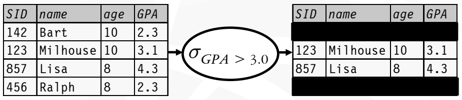
Selection

- 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	10	2.3
123	Milhouse	10	3.1
857	Lisa	8	4.3
456	Ralph	8	2.3

SID	*name*	*age*	*GPA*
[redacted]	[redacted]	[redacted]	[redacted]
123	Milhouse	10	3.1
857	Lisa	8	4.3
[redacted]	[redacted]	[redacted]	[redacted]

More on selection

- Selection predicate in general can include any column of *R*, constants, comparisons such as $=$, $\leq$, etc., and Boolean connectives $\wedge$, $\vee$, and $\neg$
  - Example: straight A students under 18 or over 21

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

    $\sigma_{\cancel{GPA > \text{all GPA in } \mathit{Student} \text{ table}}} (\mathit{Student})$

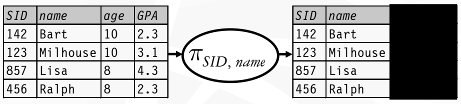
Projection

- 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

  $\pi_{SID,\, name} (\mathit{Student})$



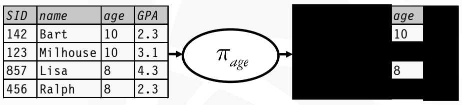
SID	*name*	*age*	*GPA*
142	Bart	10	2.3
123	Milhouse	10	3.1
857	Lisa	8	4.3
456	Ralph	8	2.3

SID	*name*	[redacted]
142	Bart	[redacted]
123	Milhouse	[redacted]
857	Lisa	[redacted]
456	Ralph	[redacted]

More on projection

- ❖ Duplicate output rows must be removed
  - ▪ Example: student ages

π_{age} (*Student*)



SID	name	age	GPA
142	Bart	10	2.3
123	Milhouse	10	3.1
857	Lisa	8	4.3
456	Ralph	8	2.3

[redacted]

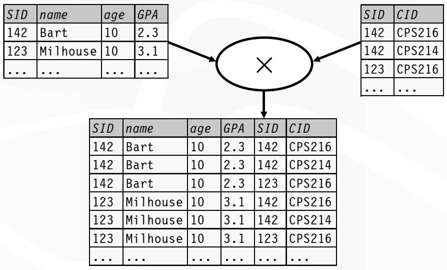
age
10
8

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	10	2.3
123	Milhouse	10	3.1
...	...	...	...

SID	CID
142	CPS216
142	CPS214
123	CPS216
...	...

SID	name	age	GPA	SID	CID
142	Bart	10	2.3	142	CPS216
142	Bart	10	2.3	142	CPS214
142	Bart	10	2.3	123	CPS216
123	Milhouse	10	3.1	142	CPS216
123	Milhouse	10	3.1	142	CPS214
123	Milhouse	10	3.1	123	CPS216
...	...	...	...	...	...

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	CPS216
142	Bart	10	2.3	142	CPS214
142	Bart	10	2.3	123	CPS216
123	Milhouse	10	3.1	142	CPS216
123	Milhouse	10	3.1	142	CPS214
123	Milhouse	10	3.1	123	CPS216
...	...	...	...	...	...

=

SID	CID	SID	name	age	GPA
142	CPS216	142	Bart	10	2.3
142	CPS214	142	Bart	10	2.3
123	CPS216	142	Bart	10	2.3
142	CPS216	123	Milhouse	10	3.1
142	CPS214	123	Milhouse	10	3.1
123	CPS216	123	Milhouse	10	3.1
...	...	...	...	...	...

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

Derived operator: join

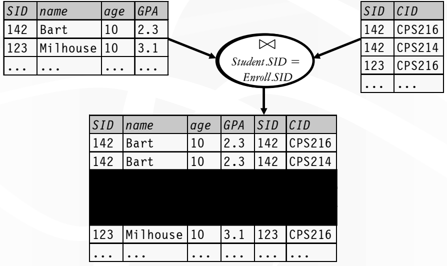
- ❖ Input: two tables R and S
- ❖ Notation: $R \bowtie_p S$
  - ▪ p is called a join condition/predicate
- ❖ Purpose: relate rows from two tables according to some criteria
- ❖ Output: for each row r in R and each row s in S, output a row rs if r and s satisfy p
- ❖ Shorthand for σ_p ($R \times S$)

Join example

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

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

Use *table.column* to disambiguate columns if necessary



SID	name	age	GPA
142	Bart	10	2.3
123	Milhouse	10	3.1
...	...	...	...

SID	CID
142	CPS216
142	CPS214
123	CPS216
...	...

SID	name	age	GPA	SID	CID
142	Bart	10	2.3	142	CPS216
142	Bart	10	2.3	142	CPS214
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
123	Milhouse	10	3.1	123	CPS216
...	...	...	...	...	...

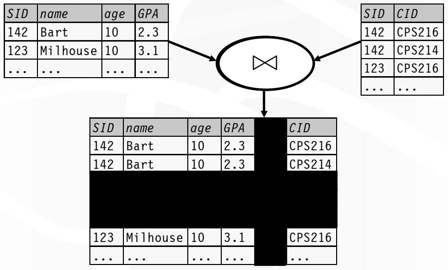
Derived operator: natural join

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

Natural join example

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

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



SID	name	age	GPA
142	Bart	10	2.3
123	Milhouse	10	3.1
...	...	...	...

SID	CID
142	CPS216
142	CPS214
123	CPS216
...	...

SID	name	age	GPA		CID
142	Bart	10	2.3	[redacted]	CPS216
142	Bart	10	2.3	[redacted]	CPS214
[redacted]					
123	Milhouse	10	3.1	[redacted]	CPS216
...	...	...	...	[redacted]	...

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 duplicates 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: $R \cap S$
  - 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 $R \bowtie S$

Renaming

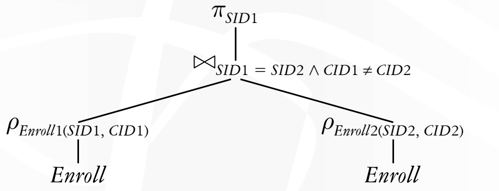
- Input: a table R
- Notation: $\rho_S (R)$, or $\rho_{S(A_1, A_2, \ldots)} (R)$
- Purpose: rename a table and/or its columns
- Output: a renamed table with the same rows as R
- Used to
  - Avoid confusion caused by identical column names
  - Create identical columns names for natural joins

Renaming example

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

$Enroll \bowtie_? Enroll$

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



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 to processing power

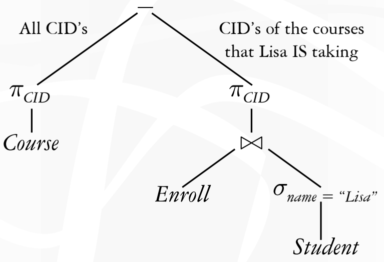
Summary of derived operators

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

- Many more
    - Semijoin, anti-semijoin, quotient, …

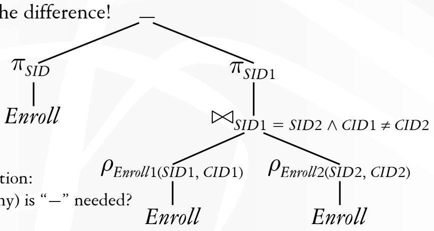
An exercise

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



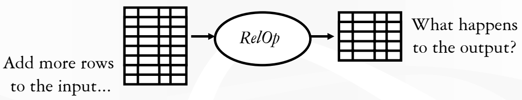
A trickier exercise

- SID's of students who take exactly one course
    - Those who take at least one course
    - Those who take at least two courses
    - Take the difference!



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

Monotone operators



- If some old output rows may be removed
    - Then the operator is non-monotone
- Otherwise the operator is monotone
    - That is, old output rows remain "correct" when more rows are added to the input
    - Formally, $R \subseteq R'$ implies $RelOp(R) \subseteq RelOp(R')$

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: $R - S$	Non-monotone (not w.r.t. S)
❖ Intersection: $R \cap S$	Monotone

Why is "—" needed for "exactly one"?

- Composition of monotone operators produces a monotone query
  - Old output rows remain "correct" when more rows are added to the input
- Exactly-one query is non-monotone
  - Say Nelson is currently taking only CPS216
  - Add another record to *Enroll*: Nelson takes CPS214 too
  - Nelson is no longer in the answer
- So it must use difference!

Why do we need core operator *X*?

- Difference
  - The only non-monotone operator
- Projection
  - The only operator that removes columns
- Cross product
  - The only operator that adds columns
- Union
  - The only operator that allows you to add rows?
  - A more rigorous proof?
- Selection? ☺

Why is r.a. a good query language?

- Declarative?
  - Yes, compared with older languages like CODASYL
  - Though operators still feel "procedural"
- Simple
  - A small set of core operators who semantics are easy to grasp
- Complete?
  - With respect to what?

Relational calculus

- $\{ e.SID \mid e \in Enroll \wedge \neg(\exists e' \in Enroll: e'.SID = e.SID \wedge e'.CID \neq e.CID)\}$ or
  $\{ e.SID \mid e \in Enroll \wedge (\forall e' \in Enroll: e'.SID \neq e.SID \vee e'.CID = e.CID)\}$
- 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

Turing machine?

- 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 recursion?
  - Optimization becomes undecidable
  - You can always implement it at the application level
  - Recursion is added to SQL nevertheless