

# Relational Model & Algebra

CPS 216  
Advanced Database Systems

## Announcements

- ❖ Homework #1 and course project will be assigned next Wednesday (January 22)
- ❖ Reading assignments in the “red book” will be photocopied and available outside my office (D327)
  - Codd paper is already there!
- ❖ There IS a recitation session this Friday (January 17) on E/R database design and review of relational algebra and design theory
- ❖ Instructor and TA office hours will be finalized this Wednesday

## 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*

<i>SID</i>	<i>name</i>	<i>age</i>	<i>GPA</i>
142	Bart	10	2.3
123	Milhouse	10	3.1
857	Lisa	8	4.3
456	Ralph	8	2.3
...	...	...	...

*Course*

<i>CID</i>	<i>title</i>
CPS216	Advanced Database Systems
CPS230	Analysis of Algorithms
CPS214	Computer Networks
...	...

*Enroll*

<i>SID</i>	<i>CID</i>
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

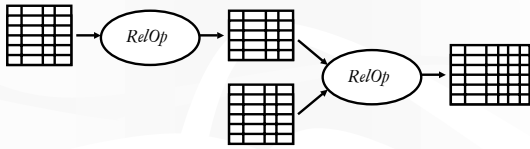
- ❖ Schema (metadata)
    - Specification of how data is to be structured logically
    - Defined at set-up
    - Rarely changes
  - ❖ Instance
    - Content
    - Changes rapidly, but always conforms to the schema
- ☞ Compare to types and instances of types in a programming language

## 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

7



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

## Selection

8

- ❖ 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

9

- ❖ Students with GPA higher than 3.0

$\sigma_{GPA > 3.0}(Student)$

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

$\sigma_{GPA > 3.0}$

<i>SID</i>	<i>name</i>	<i>age</i>	<i>GPA</i>
123	Milhouse	10	3.1
857	Lisa	8	4.3

## More on selection

10

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

~~$\sigma_{GPA > \text{all}(GPA)}(Student\ table)$~~   $\sigma_{GPA > 4.0 \wedge (age < 18 \vee age > 21)}(Student)$

## Projection

11

- ❖ 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

12

- ❖ ID's and names of all students

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

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

$\pi_{SID, name}$

<i>SID</i>	<i>name</i>
142	Bart
123	Milhouse
857	Lisa
456	Ralph

## More on projection

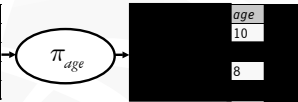
13

- ❖ Duplicate output rows must be removed

- Example: student ages

$\pi_{age}(Student)$

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



## Cross product

14

- ❖ 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

15

- ❖  $Student \times Enroll$

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

16

- ❖ The ordering of columns in a table is considered unimportant (so 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

17

- ❖ 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

18

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

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

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	SID	CID
142	Bart	10	2.3	142	CPS216
142	Bart	10	2.3	142	CPS214
...	...	...	...	...	...

## Derived operator: natural join

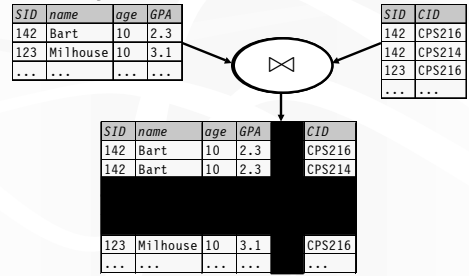
19

- ❖ 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

20

$$\begin{aligned} \text{Student} \bowtie \text{Enroll} &= \pi_{\rho} (\text{Student} \bowtie_{\rho} \text{Enroll}) \\ &= \pi_{SID, name, age, GPA, CID} (\text{Student} \bowtie_{\text{Student.SID} = \text{Enroll.SID}} \text{Enroll}) \end{aligned}$$



## Union

21

- ❖ 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

22

- ❖ 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

23

- ❖ 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

24

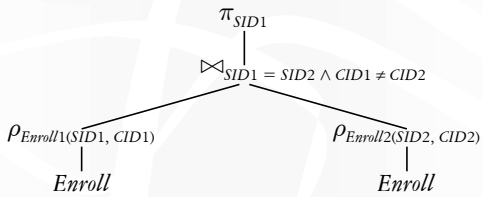
- ❖ Input: a table  $R$
- ❖ Notation:  $\rho_S (R)$ , or  $\rho_{S(A_1, A_2, \dots)} (R)$
- ❖ Purpose: rename a table and/or its columns
- ❖ Output: a renamed table with the same rows as  $R$
- ❖ Used to
  - Avoid confusion caused by identical column names
  - Create identical column names for natural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

### Renaming example

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

$Enroll \bowtie_1 Enroll$

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



### Summary of core operators

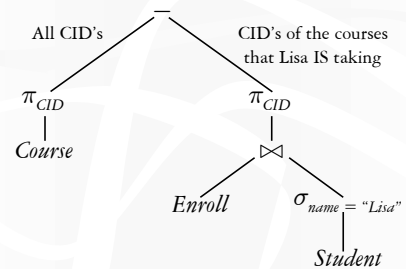
- ❖ Selection:  $\sigma_p ( R )$
- ❖ Projection:  $\pi_L ( R )$
- ❖ Cross product:  $R \times S$
- ❖ Union:  $R \cup S$
- ❖ Difference:  $R - S$
- ❖ Renaming:  $\rho_{S(A_1, A_2, \dots)} ( R )$ 
  - Does not really add to expressive power

### Summary of derived operators

- ❖ Join:  $R \bowtie_p S$
- ❖ Natural join:  $R \bowtie S$
- ❖ Intersection:  $R \cap S$
- ❖ Many more
  - Semijoin, anti-semijoin, quotient, ...

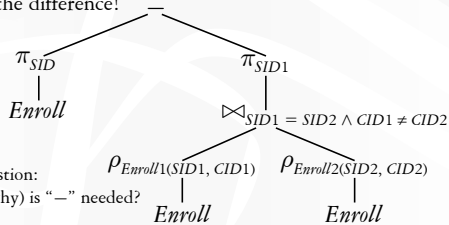
### An exercise

❖ 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 must 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 <sup>31</sup>

- ❖ 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”? <sup>32</sup>

- ❖ 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? <sup>33</sup>

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

## Why is r.a. a good query language? <sup>34</sup>

- ❖ Declarative?
  - Yes, compared with older languages like CODASYL
  - But operators are inherently procedural
- ❖ Simple
  - A small set of core operators whose semantics are easy to grasp
- ❖ Complete?
  - With respect to what?

## Relational calculus <sup>35</sup>

- ❖  $\{ s.SID \mid s \in Student \wedge \neg(\exists s' \in Student: s.GPA < s'.GPA) \}$ , 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

## Turing machine? <sup>36</sup>

- ❖ 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