Relational Model & Algebra

CompSci 316

Introduction to Database Systems

Announcements (Tue. Sep. 3)

- Homework #1 has been posted
  - Sign up for Gradiance now!
  - Windows Azure passcode will be emailed soon—sign up as soon as you can!
- Office hours: see course website
- Lecture notes
  - The "notes" version can be (printed out and) used for note-taking; the "complete" version will be posted after lecture, so be selective in what you copy down
- Readings: see Tentative Syllabus on course website
- Working on the enrollment issue
  - *If you were on the wait list before it disappeared on Friday night, email me with your last remembered position*

Relational data model

- A database is a collection of relations (or tables)
- Each relation has a list of attributes (or columns)
- Each attribute has a domain (or type)
  - Set-valued attributes not allowed
- Each relation contains a set of tuples (or rows)
  - Each tuple has a value for each attribute of the relation
  - Duplicate tuples are not allowed
    - Two tuples are identical if they agree on all attributes

☞ Simplicity is a virtue!

Example

4

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*
CPS316	Intro. to Database Systems
CPS330	Analysis of Algorithms
CPS310	Computer Networks
…	…

Enroll

SID	*CID*
142	CPS316
142	CPS310
123	CPS316
857	CPS316
857	CPS330
456	CPS310
…	…

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

Schema versus instance

5

- 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 objects of type in a programming language

Example

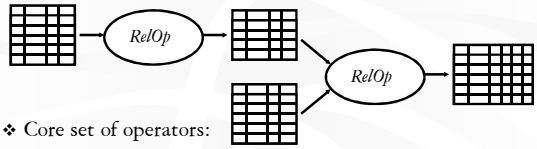
6

- 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⟩, … }
  - {⟨CPS316, Intro. to Database Systems⟩, … }
  - {⟨142, CPS316⟩, ⟨142, CPS310⟩, … }

Relational algebra

7

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

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

$\sigma_{GPA>3.0} Student$

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

$\sigma_{GPA>3.0}$

SID	name	age	GPA
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 ($=, \leq$, etc.), and Boolean connectives ($\wedge$: and, $\vee$: or, and $\neg$: not)
  - Example: straight A students under 18 or over 21

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

    $\sigma_{GPA \geq all\ GPA\ in\ Student} Student$

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

  $\pi_{SID,name} Student$

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

$\pi_{SID,name}$

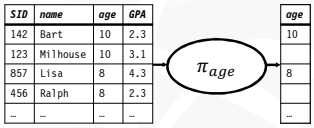
SID	name
142	Bart
123	Milhouse
857	Lisa
456	Ralph
…	…

More on projection

13

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

$\pi_{age} Student$



Cross product

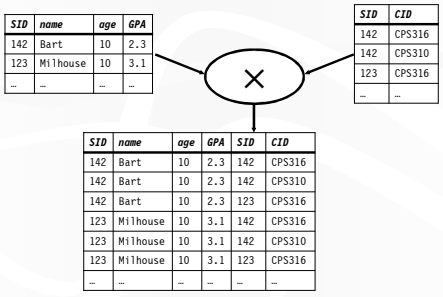
14

- 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

- $Student \times Enroll$



A note on column ordering

16

- The ordering of columns in a table is unimportant as far as the contents are concerned

SID	name	age	GPA	SID	CID
142	Bart	10	2.3	142	CPS316
142	Bart	10	2.3	142	CPS310
142	Bart	10	2.3	123	CPS316
123	Milhouse	10	3.1	142	CPS316
123	Milhouse	10	3.1	142	CPS310
123	Milhouse	10	3.1	123	CPS316
…	…	…	…	…	…

=

SID	CID	SID	name	age	GPA
142	CPS316	142	Bart	10	2.3
142	CPS310	142	Bart	10	2.3
123	CPS316	142	Bart	10	2.3
142	CPS316	123	Milhouse	10	3.1
142	CPS310	123	Milhouse	10	3.1
123	CPS316	123	Milhouse	10	3.1
…	…	…	…	…	…

- So cross product is commutative, i.e., for any R, S: $R \times S = S \times R$ (up to the ordering of output columns)

Derived operator: join

17

(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

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
…	…	…	…

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

SID	CID
142	CPS316
142	CPS310
123	CPS316
…	…

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

SID	name	age	GPA	SID	CID
142	Bart	10	2.3	142	CPS316
142	Bart	10	2.3	142	CPS310
					CPS316
					CPS316
					CPS310
123	Milhouse	10	3.1	123	CPS316
…	…	…	…	…	…

Derived operator: natural join

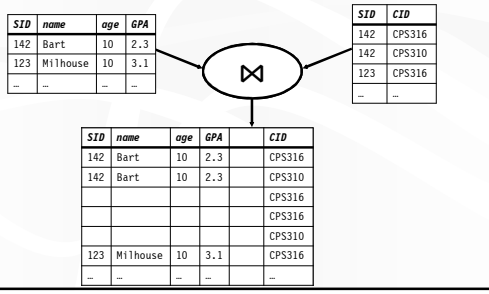
19

- 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 $\pi_L(R \bowtie_p S)$, 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

20

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



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

SID	CID
142	CPS316
142	CPS310
123	CPS316
…	…

SID	name	age	GPA		CID
142	Bart	10	2.3		CPS316
142	Bart	10	2.3		CPS310
					CPS316
					CPS316
					CPS310
123	Milhouse	10	3.1		CPS316
…	…	…	…		…

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 duplicate rows 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

Renaming

24

- 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

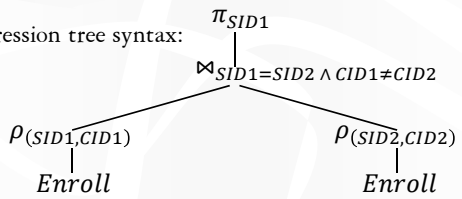
25

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

$Enroll \bowtie_? Enroll$

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

Expression tree syntax:



Summary of core operators

26

- 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 "processing" power

Summary of derived operators

27

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

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

An exercise

28

- Names of students in Lisa's classes

Writing a query bottom-up:

Their names

Students in Lisa's classes

Lisa's classes

Who's Lisa?

$\sigma_{name="Lisa"}$

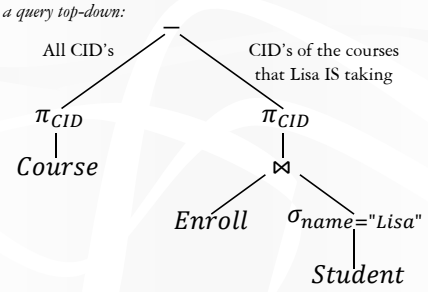
$Student$

Another exercise

29

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

Writing a query top-down:



A trickier exercise

30

- Who has the highest GPA?

Monotone operators

31

- 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

32

- Selection: $\sigma_p R$
- Projection: $\pi_L R$
- Cross product: $R \times S$
- Join: $R \bowtie_p S$
- Natural join: $R \bowtie S$
- Union: $R \cup S$
- Difference: $R - S$
- Intersection: $R \cap S$

Why is "−" needed for highest GPA?

33

- Composition of monotone operators produces a monotone query
  - Old output rows remain "correct" when more rows are added to the input
- Is highest-GPA query monotone?

Why do we need core operator *X*?

34

- Difference
- Cross product
- Union
- Selection? Projection?
  - Homework problem ☺

Extensions to relational algebra

35

- Duplicate handling ("bag algebra")
- Grouping and aggregation
- Extension (or extended projection) to allow new attribute values to be computed

- ☞ All these will come up when we talk about SQL
- ☞ But for now we will stick to standard relational algebra without these extensions

Why is r.a. a good query language?

36

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

Relational calculus

37

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

38

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