Relational Databases

1. Relational Algebra (and connection to Datalog)

Relational Databases

- ***Relational database*: a set of *relations/tables***
- ***Relation schema* : specifies**
  - *name of relation,*
  - *fixed set of* columns (a.k.a. fields, attributes) *with distinct identifiers*
  - *(eventually additional information about the columns)*

For now we continue as in logic, where arguments are identified by position. Use names to give intuition of what is in each column.

E.G.

Student(sid, name, age, gpa).

- **Relation instance:** is a *set* of tuples (rows/records) of values in the columns. Think of it as a table
  - (As a set, it is not supposed to have duplicates!)
  - Must "match" the schema

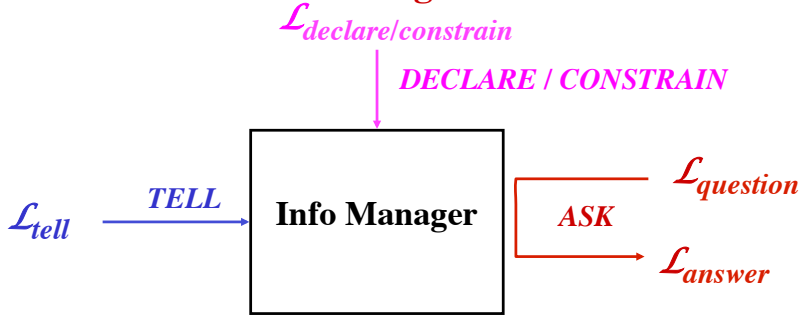
Schema & Example Instance of *Student* Relation

Student(sid,name,age,gpa)

sid	name	age	gpa
5366	Jones	18	3.4
53688	Smith	18	3.2
53650	Smith	19	3.8

- *cardinality (number of columns)* = 4, all rows distinct
- NULL value: unknown or non-existent (e.g. phone)

Extended Functional view of Information Manager



TELL*:** $\mathcal{L}_{tell} \times$ **IM** → **IM** ∪ ***Exceptions

***ASK*:** $\mathcal{L}_{question} \times$ **IM** → $\mathcal{L}_{answer}$

***DEFINE*:** $\mathcal{L}_{declare} \times$ **IM** → **IM**

Relational databases (in contrast to $Datalog_{0,5}$)

- ***tell*: ground atomic formulae as tuples in tables *(no rules!)***

```
person(bob). food(pie). likes(bob,eve).
likes(eve,pie)
```

- ***question* :**
  1. "relational algebra"
  2. SQL
  3. arbitrary First Order Logic formulae

- ***answer* : set of tuples in a table**

Relational Algebra

- **What is an Algebra?**
  - constants
  - expressions made of operators applied to operand expressions
  - laws for manipulating expressions to get *equivalent* ones
- **e.g., Algebra of sets:**
  - **start from sets defined by enumeration**
  - **operators:** union, intersection, complement
- **Algebra of Relational Tables**
  - not really used for querying by end-users
  - more like instructions to compute answers
  - but useful terminology (everyone uses it), and used in making queries run more efficiently (optimization)

"Constants"

student(Sid,Name,Age, Gpa)
course(Cid, Title, Dept)
enrolledIn(Sid,Cid,Grade)

Table name

e.g. "Find all students"

- **Algebra**

  student

- **(Datalog**

  answer(Sid,Name,Age,Gpa) :- student(Sid,Name,Age,Gpa)

"Project" operator

student(Sid,Name,Age, Gpa)
course(Cid, Title, Dept)
enrolledIn(Sid,Cid,Grade)

Eliminates columns

e.g. "Find name and age of students"

- **Algebra**

  PROJECT[#2,#3](student) | *PROJECT[Name,Age](student)*

  $\pi_{\#2,\#3}$(student) | $\pi_{Name,Age}(student)$

- **(Datalog**

  answer(Name,Age) :- student(_,Name,Age,_)

"Select" operator

student(Sid,Name,Age, Gpa)
course(Cid, Title, Dept)
enrolledIn(Sid,Cid,Grade)

Eliminates rows

e.g. "Find all information about students whose age = 18"

- **Algebra**

  SELECT[Age=18](student)

  $\sigma_{Age=18}$(student)

- **Datalog**

answer(Sid,Name,Age,Year) :- student(Sid,Name,Age,Year), Age =18

OR

answer(Sid,Name,18,Year) :- student(Sid,Name,18,Year)

sid	name	age	gpa
5366	Jones	18	3.4
53688	Smith	18	3.2
53650	Smith	19	3.2

original Student table instance

name	gpa
Jones	3.4
Smith	3.2
Smith	3.2

projection on name,gpa

Algebra: can combine things

student(Sid,Name,Age, Gpa)
course(Cid, Title, Dept)
enrolledIn(Sid,Cid,Grade)

e.g., "*Find name of 18 year old students*"

PROJECT[name] (SELECT[age=18] (student))

sid	name	age	gpa
5366	Jones	18	3.4
53688	Smith	18	3.2
53650	Smith	19	3.8

original Student table

sid	name	age	gpa
5366	Jones	18	3.4
53688	Smith	18	3.2

answer

"Product" operator

student(Sid,Name,Age, Gpa)
course(Cid, Title, Dept)
enrolledIn(Sid,Cid,Grade)

Combines relations (cartesian product in math)

e.g., "Find all possible combinations of enrolledIn and course"
(not very useful on its own)

- **Algebra**

  PRODUCT(enrolledIn,course)

  enrolledIn × course

Name conflict needs to be resolved

- **Datalog**

```
answer(Sid,Enrol_cid,Grade,Course_cid,Title,Dept) :-
   enrolledIn(Sid,Enrol_Cid,Grade),
   course(Course_cid,Title,Dept).
```


R

p	q
a1	b1
a2	b2

T

r	s	t
c1	d1	e1
c2	d2	e2
c3	d3	e3

R x T

p	q	r	s	t
a1	b1	c1	d1	e1
a1	b1	c2	d2	e2
...				
a2	b2	c3	d3	e3

Union, Intersect, Difference

student(Sid,Name,Age, Gpa)
course(Cid, Title, Dept)
enrolledIn(Sid,Cid,Grade)

Combines relations using set-operations; must have "compatible schema" (at least same number of columns and datatypes; usually same names for columns too)

"Find Sid of young or 'A' students"

- **Algebra**

  student1 := PROJECT[sid] (SELECT[Age<18](students))
  student2 := PROJECT[sid] (SELECT[grade='A'](enrolledIn))
  s3 := UNION(student1,student2)

- **Datalog**

```
answer(Sid):- student(Sid,Name,Age,Year), Age < 18.
answer(Sid):- enrolledIn(Sid,_,'A').
```


" Difference"

student(Sid,Name,Age, Gpa)
course(Cid, Title, Dept)
enrolledIn(Sid,Cid,Grade)

e.g. "Students not taking classes"

- **Algebra**

  DIFF(PROJ[Sid](student) , PROJ(Sid)[enrolledIn))

  $\pi_{Sid}(student) \quad - \quad \pi_{Sid}(enrolledIn)$

- **Datalog**

```
answer(Sid):- student(Sid,_,_,_),
                 NOT enrolledIn(Sid,_,_)
```

What if you wanted their name too?

"Rename table and/or column"

student(Sid,Name,Age, Gpa)
course(Cid, Title, Dept)
enrolledIn(Sid,Cid,Grade)

(Trivial name change. But will be necessary when combination of multiple tables results in conflicting names)

e.g. "What are scores gotten by students taking classes."

- **Algebra**

  RENAME[taking(Grade~> Score)] (enrolledIn)

OR

  taking := RENAME[Grade~> Score] (enrolledIn)

  $\rho_{\textbf{taking(Grade~> Score)}}$ (enrolledIn)

- **Datalog**

```
taking(Sid,Cid,Score):-
  enrolledIn(Sid,Cid,Grade),Grade=Score.
```


student(Sid,Name,Age, Gpa)
course(Cid, Title, Dept)
enrolledIn(Sid,Cid,Grade)

- **Try *"Pairs of student Ids, taking the same course"***

"Natural Join" (*very useful*)

student(Sid,Name,Age, Gpa)
course(Cid, Title, Dept)
enrolledIn(Sid,Cid,Grade)

Schema: *set of attributes in both schemas R and S* (no duplics)
Tuples: *t such that projection onto R and S yield tuples in R and S*

e.g., "Find courses and grades taken by students"

- **Algebra**

  JOIN(student,enrolledIn)

  student |><| enrolledIn

- **Datalog**

```
answer(Sid,Name,Age,Year,Cid,Grade):-
  student(Sid,Name,Age,Year),
   enrolledIn(Sid,Cid,Grade)
```


students

sid	name	age	gpa
53666	Jones	18	3.4
53688	Smith	18	3.2
53650	Smith	19	3.8

enrolledIn

sid	cid	grade
53666	205	A
53666	206	F

JOIN[*students,enrolledIn*]

sid	name	age	gpa	cid	grade
53666	Jones	18	3.4	205	A
53666	Jone	18	3.4	206	F

Combine rows from S and E where columns with the same name have identical values, and eliminate duplicate columns.

Algebraic Identity

JOIN(student,enrolledIn)

=

PROJECT[Sid,Name,Age,Year,Cid,Grade]
(SELECT[Sid=Sid_t]
(PRODUCT(student,
RENAME[Sid->Sid_t](enrolledIn))))

"Division" [*tricky]

- Not supported as a primitive operator, but useful for expressing queries like:
  Find students who have enrolled in all courses.
- Let A have 2 fields, x and y; B have only field y:
  - $A/B = \{\langle x\rangle \mid \exists \langle x,y\rangle \in A \ \ \forall \langle y\rangle \in B\}$
  - i.e., ***A/B* contains all *x* tuples (*students*) such that for *every y* tuple (*course*) in *B*, there is an *xy* tuple in *A* (*enrol*).**
  - *Or*: If the set of y values (*course*) associated with an x value (*student*) in A contains all y values in B, the x value is in A/B.
- In general, x and y can be any lists of fields; y is the list of fields in B, and $x \cup y$ is the list of fields of A.

Examples of Division A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

sno
s1
s2
s3
s4

A/B1

pno
p2
p4

B2

sno
s1
s4

A/B2

pno
p1
p2
p4

B3

sno
s1

A/B3

The following examples are from textbook and were not covered in class.

Schema for the following queries:

```
sailors(sid,sname,rating,age)
boats(bid,bnam,color)
reserves(sid,bid,day)
```

"Find names of sailors who reserved boat #103"

Solution 1: $\pi_{sname}((\sigma_{bid=103}\text{Re}serves) \bowtie Sailors)$

❖ Solution 2: $\rho\ (Temp1,\ \sigma_{bid=103}\text{Re}serves)$

$\rho\,(Temp2, Temp1 \bowtie Sailors)$

$\pi_{sname}\,(Temp2)$

❖ Solution 3: $\pi_{sname}(\sigma_{bid=103}(\text{Re}serves \bowtie Sailors))$

"Find names of sailors who've reserved a red boat"

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

$\pi_{sname}((\sigma_{color='red'}Boats) \bowtie \text{Re}serves \bowtie Sailors)$

❖ A more efficient solution:

$\pi_{sname}(\pi_{sid}((\pi_{bid}\sigma_{color='red'}Boats) \bowtie \text{Re}s) \bowtie Sailors)$

A query optimizer can find this, given the first solution!

"Find sailors who've reserved a red or a green boat"

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

$\rho\,(Tempboats,(\sigma_{color='red' \vee color='green'}Boats))$

$\pi_{sname}(Tempboats \bowtie \text{Re}serves \bowtie Sailors)$

❖ Can also define Tempboats using union! (How?)

❖ What happens if $\vee$ is replaced by $\wedge$ in this query?

"Find sailors who've reserved a red and a green boat"

- **Previous approach won't work! Must identify sailors who've reserved red boats, sailors who've reserved green boats, then find the intersection (note that *sid* is a key for Sailors):**

$\rho\,(Tempred, \pi_{sid}((\sigma_{color='red'}Boats) \bowtie \text{Re}serves))$

$\rho\,(Tempgreen, \pi_{sid}((\sigma_{color='green'}Boats) \bowtie \text{Re}serves))$

$\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$

"Find the names of sailors who've reserved all boats"

- **Uses division; schemas of the input relations to division must be carefully chosen:**

$$\rho\,(Tempsids, (\pi_{sid,bid}\text{Re}serves) / (\pi_{bid}Boats))$$

$$PROJ[sname](TempsidsJOINSailors)$$

- To find sailors who've reserved all 'Interlake' boats:

$$..... \quad / \pi_{bid}(\sigma_{bname='Interlake'}Boats)$$