Relational Algebra

Introduction to Databases

Sina Meraji

Thanks to Ryan Johnson, John Mylopoulos, Arnold Rosenbloom and Renee Miller for material in these slides

Why the relational model?

- Sounds good: matches how we think about data
- Real reason: *data independence*!
- Earlier models tied to physical data layout
  - Procedural access to data (low-level, explicit access)
  - Relationships stored in data (linked lists, trees, etc.)
  - Change in data layout => application rewrite
- Relational model
  - Declarative access to data (system optimizes for you)
  - Relationships specified by queries (schemas help, too)
  - Develop, maintain apps and data layout separately

Similar battle today with languages

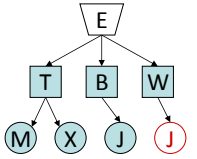
Comparing data models

Student job example

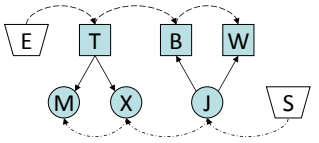
Mary (M) and Xiao (X) both work at Tim Hortons (T)

Jaspreet (J) works at both Bookstore (B) and Wind (W)

Hierarchical (tree)



Network (graph)



Relational (table)

E

B	...
T	...
W	...

S

J	...
M	...
X	...

R

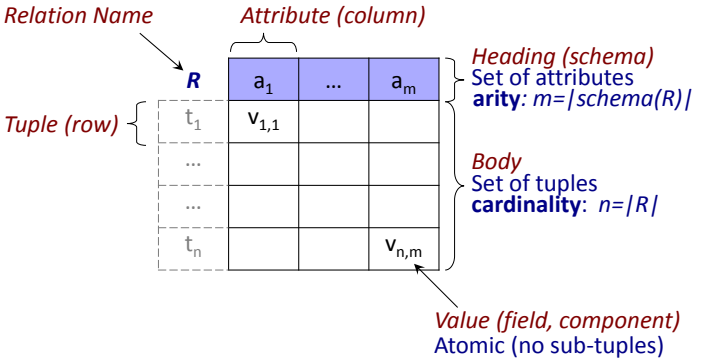
M	T
X	T
J	B
J	W

What is the relational model?

- Logical representation of data
  - Two-dimensional tables (relations)
- Formal system for manipulating relations
  - Relational algebra
- Result
  - High-level (logical, declarative) description of data
  - Mechanical rules for rewriting/optimizing low-level access
  - Formal methods to reason about soundness

Relational algebra is the key

Relations and tuples



Set-based: arbitrary row/col ordering
*Logical: physical layout might be *very* different!*

What is an algebra?

- Operands (values)
  - Variables, constants
  - Closed domain
- Operators
  - + "Addition"
  - * "Multiplication"
- Expressions:
  - Combine operations with parenthesis (explicit)
  - OR using either precedence (implied)
- Laws
  - Identify semantically equivalent expressions
  - Commutativity, associativity, etc.

Offers formal, sound, mechanical rewriting

Example algebra: integer arithmetic

- Domain: integers
  - … -100, … -1, 0, 1, … 100, …
- Operators: -, +, *, …
- Expressions
  - ((2*a) + ((5*(c + (-d))) + e))
- Laws
  - a*b = b*a
  - a*(b*c) = (a*b)*c
  - a*(b+c) = a*b + a*c

'-' is unary negation

Commutative
Associative
Distributive

Allows compilers to reason about, optimize

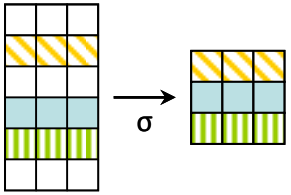
Relational algebra

- Values
  - Finite relations (cardinality and arity both bounded)
  - Attributes may or may not be typed
- Operators
  - Unary: σ, π, ρ
  - "Additive" (set): ∪, ∩, -
  - "Multiplicative:" ×, ⋈
  - [details to come]
- Expressions
  - Same as arithmetic, but called "queries"
- Laws
  - Allow "query rewriting"
  - Basis for query optimization
  - [details to come]

Expressive power equivalent to 1st order logic

Unary operators: select (σ)

- $\sigma_P(R)$ outputs tuples of R which satisfy P
- same schema as R



Removes unwanted rows from relation

Unary operators: select (σ) example

Employees

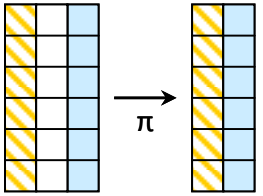
Surname	FirstName	Age	Salary
Smith	Mary	25	2000
Black	Lucy	40	3000
Verdi	Nico	36	4500
Smith	Mark	40	3900

$\sigma_{Age<30\ V\ Salary>4000}$ (Employees)

Surname	FirstName	Age	Salary
Smith	Mary	25	2000
Verdi	Nico	36	4500

Unary operators: project (π)

- $\pi_Y(R)$ outputs a subset Y of the set of attributes X of relation R



Removes unwanted columns from relation

Unary operators: project (π) example

Employees

Surname	FirstName	Department	Head
Smith	Mary	Sales	De Rossi
Black	Lucy	Sales	De Rossi
Verdi	Mary	Personnel	Fox
Smith	Mark	Personnel	Fox

$\pi_{Surname,\ FirstName}$(Employees)

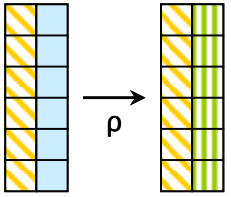
Surname	FirstName
Smith	Mary
Black	Lucy
Verdi	Mary
Smith	Mark

$\pi_{Department,\ Head}$ (Employees)

Department	Head
Sales	De Rossi
Personnel	Fox

Unary operators: rename (ρ)

- $\rho_{S(A,B,C)}(R)$ renames attributes of R to A,B,C and calls the result S
  - $\rho_S(R)$ renames relation R (same attributes)
  - $\rho_{A=X,C=Y}(R)$ (or $\rho_{A,\ C \to X,\ Y}(R)$) renames attributes A and C only



Modifies schema only - same values

Unary operators: rename (ρ) example

Paternity

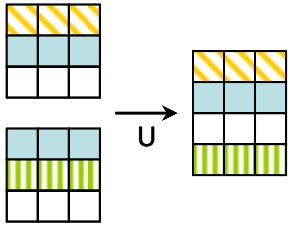
Father	Child
Adam	Cain
Adam	Abel
Abraham	Isaac
Abraham	Ishmael

$\rho_{Father \to Parent}$(Paternity)

Parent	Child
Adam	Cain
Adam	Abel
Abraham	Isaac
Abraham	Ishmael

Additive operators (∪, ∩, -)

- Standard set operators
- Operate on tuples within input relations, but not on schema



Additive operators: Union (U)

Graduates

Number	Surname	Age
7274	Robinson	37
7432	O'Malley	39
9824	Darkes	38

Managers

Number	Surname	Age
9297	O'Malley	56
7432	O'Malley	39
9824	Darkes	38

Graduates ∪ Managers

Number	Surname	Age
7274	Robinson	37
7432	O'Malley	39
9824	Darkes	38
9297	O'Malley	56

Additive operators: Intersection (∩)

Graduates

Number	Surname	Age
7274	Robinson	37
7432	O'Malley	39
9824	Darkes	38

Managers

Number	Surname	Age
9297	O'Malley	56
7432	O'Malley	39
9824	Darkes	38

Graduates ∩ Managers

Number	Surname	Age
7432	O'Malley	39
9824	Darkes	38

Additive operators: Difference (-)

Graduates

Number	Surname	Age
7274	Robinson	37
7432	O'Malley	39
9824	Darkes	38

Managers

Number	Surname	Age
9297	O'Malley	56
7432	O'Malley	39
9824	Darkes	38

Graduates - Managers

Number	Surname	Age
7274	Robinson	37

A Meaningful but Impossible Union

Paternity

Father	Child
Adam	Cain
Adam	Abel
Abraham	Isaac
Abraham	Ishmael

Maternity

Mother	Child
Eve	Cain
Eve	Seth
Sarah	Isaac
Hagar	Ishmael

Paternity ∪ Maternity ???

- The problem: Father and Mother are different names, but both represent a parent
- Solution: rename attributes!

Union with Renaming

Paternity

Father	Child
Adam	Cain
Adam	Abel
Abraham	Isaac
Abraham	Ishmael

Maternity

Mother	Child
Eve	Cain
Eve	Seth
Sarah	Isaac
Hagar	Ishmael

$\rho_{Father->Parent}(Paternity) \cup \rho_{Mother->Parent}(Maternity)$

Parent	Child
Adam	Cain
Adam	Abel
Abraham	Isaac
Abraham	Ishmael
Eve	Cain
Eve	Seth
Sarah	Isaac
Hagar	Ishmael

Union with Renaming (Many Attributes)

Employees

Surname	Branch	Salary
Patterson	Rome	45
Trumble	London	53

Staff

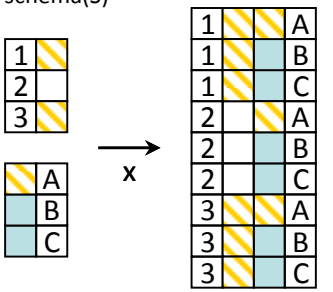
Surname	Factory	Wages
Cooke	Chicago	33
Bush	Monza	32

$\rho_{Branch,Salary \to Location,Pay}(Employees) \cup \rho_{Factory,\ Wages \to Location,Pay}(Staff)$

Surname	Location	Pay
Patterson	Rome	45
Trumble	London	53
Cooke	Chicago	33
Bush	Monza	32

Cartesian product (×)

- The outcome of combining every record in R with every record in S
- T = R × S contains every pairwise combination of R and S tuples
  - schema(T) = schema(R) U schema(S)
  - |T| = |R|*|S|



*Input schemas must *not* overlap*

Cartesian product (×) example

Employees

Employee	Project
Smith	A
Black	A
Black	B

Projects

Code	Name
A	Venus
B	Mars

Employees × Projects

Employee	Project	Code	Name
Smith	A	A	Venus
Black	A	A	Venus
Black	B	A	Venus
Smith	A	B	Mars
Black	A	B	Mars
Black	B	B	Mars

Division (/)

- Let R and S be relations with schemas A1, …, An, B1, …, Bn and B1, …, Bn respectively. The result of R/S is a relation T with
  - Schema A1, …, An (*attribute names in R but not in S*)
  - Tuples t such that, for every tuple s of S, the tuple t||s (the concatenation of t and s) is in relation R
  - T contains the largest possible set of tuples s. t. S × T ⊆ R
- Analogy to integer division:
  - For integers, A / B is: the largest int Q s.t. Q x B ≤ A
  - For relations, A / B is: the largest relation Q s.t. Q × B ⊆ A

Division example

R

A	B	C
a1	b1	c1
a2	b1	c1
a1	b2	c1
a1	b2	c2
a2	b1	c2
a1	b2	c3
a1	b2	c4
a1	b1	c5

S1

C
c1

→

A	B
a1	b1
a2	b1
a1	b2

T1=R/S1

S2

C
c1
c2

→

A	B
a1	b2
a2	b1

T2=R/S2

S3

C
c1
c2
c3
c4

→

A	B
a1	b2

T3=R/S3

Division example (cont.)

R

A	B	C
a1	b1	c1
a2	b1	c1
a1	b2	c1
a1	b2	c2
a2	b1	c2
a1	b2	c3
a1	b2	c4
a1	b1	c5

S4

B	C
b1	c1

→

A
a1
a2

T4=R/S4

S5

B	C
b1	c1
b2	c1

→

A
a1

T5=R/S5

Division in RA

- Consider two relations A(x,y), B(y) and suppose we want to specify the query

  "Find all x's that are associated through A with all B's"

  – This can be expressed as:

  $$A/B = \pi_x(A) - \pi_x((\pi_x(A) \times B) - A)$$

  – Often useful when the query is about “every” or “all” (but don’t just look for these keywords!)

  – Doesn’t extend the expressiveness of Relational Algebra (convenient to use in many situations)

Division in RA example

- Assume
  - Take(x,y) - "student x has taken course y",
  - CS(z) - "z is a CS course"
- We want "All students who have taken all CS courses"
  - $\pi_x(\text{Take}) \times \text{CS}$
    (Relation of all <student, CS course> pairs)
  - $(\pi_x(\text{Take}) \times \text{CS}) - \rho_{y->z}(\text{Take})$
    (Relation of all <student, CS course> pairs that did NOT occur)
  - $\pi_x((\pi_x(\text{Take}) \times \text{CS}) - \rho_{y->z}(\text{Take}))$
    (Relation of all <students> who have NOT taken all CS courses)
  - $\pi_x(\text{Take}) - \pi_x((\pi_x(\text{Take}) \times \text{CS}) - \rho_{y->z}(\text{Take}))$
    (Relation of all <students> who have taken all CS courses)

Work a simple example at home

Division Example

Take

x	y
S1	C1
S1	C2
S1	C3
S2	C1
S3	C1
S3	C3
S4	C1
S4	C2
S4	C3
S5	C2

CS

z
C1
C2
C3

R1

x	z
S1	C1
S1	C2
S1	C3
S2	C1
S2	C2
S2	C3
S3	C1
S3	C2
S3	C3
S4	C1
S4	C2
S4	C3
S5	C1
S5	C2
S5	C3

R2

x	z
S2	C2
S2	C3
S3	C2
S5	C1
S5	C3

R3

x
S2
S3
S5

R4

x
S1
S4

$\pi_x(\text{Take}) - \pi_x((\pi_x(\text{Take}) \times \text{CS}) - \rho_{y->z}(\text{Take}))$

Join

- The most used operator in relational algebra
- Used to establish connections among data in different relations, taking advantage of the "value-based" nature of the relational model
- Two main versions of the join:
  - natural join: takes **attribute names** into account
  - theta join: takes **attribute values** into account
- Both join operations denoted by the symbol ⋈

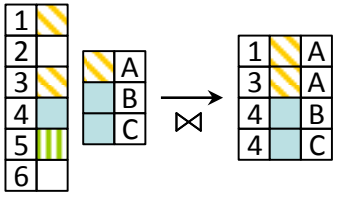
Natural join (⋈)

- T = R ⋈ S merges tuples from R and S having equal values where their schemas overlap (join attributes)
  - T Schema: Union of schemas
  - $|T| \leq |R|*|S|$, usually $\approx \max(|R|, |S|)$

schema(R) ∩ schema(S) ≠ ∅

"join cardinality"

- Special cases
  - No schema overlap: ×
  - Full schema overlap: ∩



Equivalent to π(σ(R × ρ(S)))

Natural join (⋈) example

r_1

Employee	Department
Smith	sales
Black	production
White	production

r_2

Department	Head
production	Mori
sales	Brown

$r_1 \bowtie r_2$

Employee	Department	Head
Smith	sales	Brown
Black	production	Mori
White	production	Mori

Properties of Natural Join

- Commutative:

  $R \bowtie S = S \bowtie R$
- Associative:

  $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
- N-ary joins without ambiguity:

  $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$

Example of N-ary Join Operation

r_1

Employee	Department
Smith	sales
Black	production
Brown	marketing
White	production

r_2

Department	Division
production	A
marketing	B
purchasing	B

r_3

Division	Head
A	Mori
B	Brown

$r_1 \bowtie r_2 \bowtie r_3$

Employee	Department	Division	Head
Black	production	A	Mori
Brown	marketing	B	Brown
White	production	A	Mori

Theta join

- Written as $T = R \bowtie_\theta S$
  - Outputs pairwise combinations of tuples which satisfy θ
  - Join cardinality: $|T| \leq |R| * |S|$
- Most general join
  - Arbitrary join predicate (not just equality)
- Equivalent to $\sigma_\theta(R \times \rho(S))$
  - Schemas must not overlap
  - Does not project away any attributes

Theta join example

Car

Car	CarPrice
CarA	20000
CarB	30000
CarC	50000

Boat

Boat	BoatPrice
BoatA	10000
BoatB	40000
BoatC	60000

$Car \bowtie_{CarPrice>BoatPrice} Boat$

Car	CarPrice	Boat	BoatPrice
CarA	20000	BoatA	10000
CarB	30000	BoatA	10000
CarC	50000	BoatA	10000
CarC	50000	BoatB	40000

Equijoin

- Special case of theta join
- Written as $R \bowtie_{A=X,B=Y,\ldots} S$
  - Attribute names in R and S can differ
  - Still compare values for equality
- Like natural join, but using arbitrary attributes
  - Very common due to *foreign keys* in relations
- Equivalent to $R \bowtie \rho(S)$

Equijoin example

Employees

Employee	Project
Smith	A
Black	A
Black	B

Projects

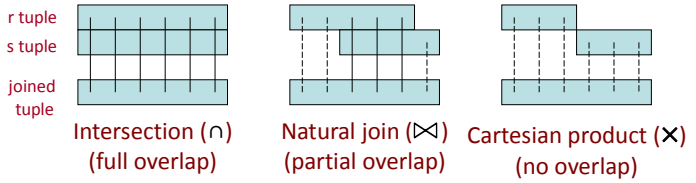
Code	Name
A	Venus
B	Mars

Employees $\bowtie_{Project=Code}$ Projects

Employee	Project	Code	Name
Smith	A	A	Venus
Black	A	A	Venus
Black	B	B	Mars

Comparison: × vs. ∩ vs. ⋈

- Same general operation
  - Test "overlapping" parts of tuples for equality
  - Combine "matching" pairs (ignore others)
- Differ in degree of schema overlap



"Generalized intersection"

Mathematical power vs. efficiency

- Note that × expresses both ∩ and ⋈
  => Mathematically, intersection and joins are unnecessary
- Why bother with them? Two big reasons
- Notation
  - $\pi(\sigma(R \times \rho(S)))$ vs. $R \cap S$
  - Cartesian product seldom useful *Why not?*
- *Performance*
  - Efficient algorithms compute result directly
  => $|R|*|S|$ rows vs. $\min(|R|,|S|)$ *Consider $|R|=|S|=10^6$*

Summary of Operators

Operation	Name	Symbol	Precedence
choose rows	select	σ	1
choose columns	project	π	
rename relation/attribute	rename	ρ	
combine tables	natural join	⋈	2
	theta join	$\bowtie_{condition}$	
	cartesian product	×	
set operations	intersection	∩	3
	union	∪	
	subtraction	−	
assignment	assignment	:=	-

Expressing Integrity Constraints

- Our text (sec 2.5) defines two ways to express an integrity constraint in Relational Algebra. Suppose R and S are expressions in RA. We can write an IC in either of these ways:
  - R = ∅ (expresses the fact that R is the empty set.)
  - R ⊆ S (expresses the fact that R is a subset of S.)
- Equivalent (we don't need the second form) but it's convenient:
  - Saying R = ∅ is equivalent to saying R ⊆ ∅.
  - Saying R ⊆ S is equivalent to saying R - S = ∅.

Expressing ICs Examples

Course (Dept, CourseNum, Title, Credits)
Section (CRN, Dept, CourseNum, Room, Time, InstructorID)

- Referential integrity constraints

  Dept and CourseNum form a foreign key within Section

  $\pi_{Dept,CourseNum}$ (Section) ⊆ $\pi_{Dept,CourseNum}$(Course)
- Key constraints

  Two tuples which agree on CRN must also agree on Dept, CourseNum, Room, Time, and InstructorID. One of the constraints implied is:

  (ρ_{S1}(Section) ⋈ $_{(S1.CRN=S2.CRN \text{ and } S1.Dept!=S2.Dept)}$ ρ_{S2}(Section)) = ∅
- Domain constraints

  E.g., "Course Numbers must be in the range 100-999"

  $\sigma_{CourseNum<100 \text{ or } CourseNum>999}$ (Course) = ∅

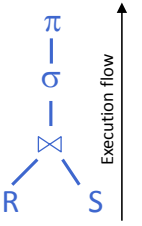
Coming next…

- Working Examples
- Tips and Tricks
- You
  - Attend the lectures
  - Read the material and practice
  - Assignment 1

TIPS & TRICKS FOR RELATIONAL ALGEBRA QUERIES

Evaluating R.A. Queries

- R.A. is procedural
  - an R.A. query itself suggests a procedure for constructing the result (i.e., how to implement the query)
- R.A. suggests a query execution plan
  - Two RA expressions might yield the same result but suggest different query execution plans
  - which is best depends on the relation cardinality, defined indices, join ordering, etc.
- DBMSs: query optimization takes place
  - Optimizer rewrites queries to be more efficient
  - topic in csc443



Tips and Tricks for R.A.

- Ask yourself which relations need to be involved?
  - Ignore the rest
- Every time you combine relations confirm that:
  - (a) attributes that should match will be made to match
  - (b) attributes that will be made to match should match

Tips and Tricks for R.A. (cont.)

- Is there an intermediate relation that would help you get the final answer?
  - Draw it out with actual data in it
- Break the answer down by defining intermediate relations using assignment:
  - Use good names for the new relations
  - Name the attributes on the Left-Hand-Side each time, so you don't forget what you have in hand
  - Add a comment that explains exactly what the relation contains

Tips for Specific R.A. Queries

- To show "max" (min is analogous):
  - Pair tuples (self-join) and find those that are not the max
  - Then substract from all to find the max[es]
- To show "k or more":
  - Make all combos of k different tuples that meet the required condition
- To show "exactly k":
  - Find "k or more"
  - Find "(k+1) or more"
  - Then subtract "(k+1) or more" from "k or more"
- To show "every" (i.e., division):
  - Make all combos that could have occurred
  - Subtract those that did occur to find those that didn't always; these are the failures
  - Subtract the failures from all to get the answer

WORKING EXAMPLES (SET-SEMANTICS)

A Sample Database

Employees

Number	Name	Age	Salary
101	Mary Smith	34	40
103	Mary Bianchi	23	35
104	Luigi Neri	38	61
105	Nico Bini	44	38
210	Marco Celli	49	60
231	Siro Bisi	50	60
252	Nico Bini	44	70
301	Steve Smith	34	70
375	Mary Smith	50	65

Supervision

Head	Emp
210	101
210	103
210	104
231	105
301	210
301	231
375	252

In fact, only the database schema and integrity constraints are needed:

Employees(Number,Name,Age,Salary)
Supervision(Head,Emp)
$\pi_{Head}(Supervision) \subseteq \pi_{Number}(Employees)$
$\pi_{Emp}(Supervision) \subseteq \pi_{Number}(Employees)$

Example 1

"Find the numbers, names and ages of employees earning more than 40K"

Employees(Number,Name,Age,Salary)
Supervision(Head,Emp)

$\pi_{Number,Name,Age}(\sigma_{Salary>40}Employees)$

Number	Name	Age
104	Luigi Neri	38
210	Marco Celli	49
231	Siro Bisi	50
252	Nico Bini	44
301	Steve Smith	34
375	Mary Smith	50

Example 2

"Find the registration numbers of the supervisors of the employees earning more than 40K."

Employees(Number,Name,Age,Salary)
Supervision(Head,Emp)

$\pi_{Head}(Supervision \bowtie_{Emp=Number}(\sigma_{Salary>40}Employees))$

Head
210
301
375

R(Head,Emp,Number,Name,Age,Salary)

Example 3

"Find the names and salaries of the supervisors of the employees earning more than 40K."

Employees(Number,Name,Age,Salary)
Supervision(Head,Emp)

$\pi_{NameH,SalH}(\rho_{Number,Name,Salary,Age\rightarrow NumH,NameH,SalH,AgeH}Employees \bowtie_{NumberH=Head} (Supervision \bowtie_{Number=Emp}(\sigma_{Salary>40}Employees)))$

NameH	SalaryH
Marco Celli	60
Steve Smith	70
Mary Smith	65

R(NumH,NameH,AgeH,SalH,Head,Emp,Number,Name,Age,Salary)

Example 4

"Find the employees earning more than their respective supervisors; return registration numbers, names and salaries of the employees and their supervisors."

Employees(Number,Name,Age,Salary)
Supervision(Head,Emp)

$\pi_{Number,Name,Salary,NumH,Nam3H,\ SalH}(\sigma_{Salary>SalH}(\rho_{Number,Name,Salary,Age\rightarrow NumH,NameH,SalH,AgeH}Employees \bowtie_{NumH=Head} (Supervision \bowtie_{Emp=Number} Employees)))$

Number	Name	Salary	NumH	NameH	SalH
104	Luigi Neri	61	210	Marco Celli	60
252	Nico Bini	70	375	Mary Smith	65

R(NumH,NameH,AgeH,SalH,Head,Emp,Number,Name,Age,Salary)

Example 5

"Find registration numbers and names of supervisors, *all* of whose employees earn more than 40K."

Employees(Number,Name,Age,Salary)
Supervision(Head,Emp)

$\pi_{Number,Name}(Employees \bowtie_{Number=Head} (\pi_{Head}(Supervision) - \pi_{Head}(Supervision \bowtie_{Number=Emp} (\sigma_{Salary\leq 40}Employees))))$

Number	Name
301	Steve Smith
375	Mary Smith

Example 6

"Find registration numbers of supervisors, who supervise *all* employees earning more than 40K."

Employees(Number,Name,Age,Salary)
Supervision(Head,Emp)

π_{Number} (Supervision / $\pi_{Number}(\sigma_{Salary>40}$Employees))

Example 7

"Find the employees earning **maximum** salary."

Employees(Number,Name,Age,Salary)
Supervision(Head,Emp)

$A = \pi_{Salary}(Emp) - \pi_{Salary}(\sigma_{Salary1>Salary}(Emp \times \rho_{everything}(Emp))$

A provides the maximum salary; the rest is easy …
$\pi_{Number}(Emp \bowtie A)$

- How to find **minimum** salary?
- How to find **average** salary?

Example 8

"Find all locations that have **at least two** employees earning more than 40K"

Employees(Number,Name,Loc,Salary)
Supervision(Head,Emp)

Assume that in the schema we have '**Loc**' instead of '**Age**' for this example only.

$\sigma_{salary>40}(Emp) \bowtie \sigma_{salary1>40}(\rho_{Number,Name,Salary \rightarrow Num1,Name1,Sal1}Emp)$

This results in a relation
R(Number, Name, Salary, Loc, Num1, Name1, Sal1)

Now we select tuples where Number≠Num1 and project on Loc
$\pi_{Loc}(\sigma_{Number \neq Num1}R)$

- How to find locations that have **at least three** employees …?
- How to find locations that have **exactly two** employees …?

Another Series of Examples

Films(Film#, Title, Director, Year, ProdCost)
Artists(Actor#, Surname, FirstName, Sex, Birthday, Nationality)
Roles(Film#, Actor#, Character)

Example 1

Films(Film#, Title, Director, Year, ProdCost)
Artists(Actor#, Surname, FirstName, Sex, Birthday, Nationality)
Roles(Film#, Actor#, Character)

"Find the titles of films starring Henry Fonda"

$\pi_{Title}(Films \bowtie (\sigma_{(FirstName="Henry") \wedge (Surname="Fonda")}(Artists \bowtie Roles)))$

Example 2

Films(Film#, Title, Director, Year, ProdCost)
Artists(Actor#, Surname, FirstName, Sex, Birthday, Nationality)
Roles(Film#, Actor#, Character)

"Find the titles of all films in which the director is also an actor"

$\pi_{Title}(\sigma_{(Director=Actor\#)}(Films \bowtie Roles))$

Example 3

Films(Film#, Title, Director, Year, ProdCost)
Artists(Actor#, Surname, FirstName, Sex, Birthday, Nationality)
Roles(Film#, Actor#, Character)

"Find the actors who have played two characters in the same film; show the title of each such film, first name and surname of the actor and the two characters"

$\pi_{Title,FirstName,Surname,Character1,Character}$
$(\rho_{Film\#,Actor\#,Character \rightarrow Film\#1,Actor\#1,Character1}(Roles)$
$\bowtie_{(Film\#1=Film\#) \wedge (Actor\#1=Actor\#) \wedge (Character1 \neq Character)}$
$Roles \bowtie Artists \bowtie Films)$

Example 4

Films(Film#, Title, Director, Year, ProdCost)
Artists(Actor#, Surname, FirstName, Sex, Birthday, Nationality)
Roles(Film#, Actor#, Character)

"Find the titles of the films in which the actors are all of the same sex"

$\pi_{Title}(Films) -$
$\pi_{Title}(Films \bowtie \sigma_{sex \neq sex1 \wedge Film\#1 = Film\#}((Artists \bowtie$
$Roles) \bowtie \rho_{Ac\#,Char,Sur,Fir,Sex,BD,N}$
$\rightarrow Ac\#1,Char1,Sur1,Fir1,Sex1,BD1,N1(Artists \bowtie Roles))$

RELATION OPERATIONS ON BAGS

Limitations of relational algebra

- Relational algebra is set-based
- Real-life applications need more
  - Expensive (and often unnecessary) to eliminate duplicates
  - Important (and often expensive) to order output
  - Need a way to apply scalar expressions to values
  - What's *not* there often as important as what is

Answer: non-set extensions

Extension: bag semantics

- In practice, relations are bags (multisets)
  - Members are allowed to appear more than once
  - Sometimes people purposefully insert duplicates
  - Projections produce duplicates
- Example: {1,2,1,1,3} is a bag (still unordered!)
- Most operators still work
  - Select, Rename remain unchanged
  - Project no longer eliminates duplicates
  - Set operations need tweaks
  - Joins tend to multiply the number of duplicates
- Some laws no longer apply

Bag versions of set operations

- Union
  - Concatenation (except unordered)
  - $\{1, 1, 2, 3\} \cup \{2, 2, 3, 4\} = \{1, 1, 2, 3, 2, 2, 3, 4\}$
- Intersection
  - Take minimum count of each value
  - $\{1, 1, 2, 3\} \cap \{2, 2, 3, 4\} = \{2, 3\}$
- Difference
  - Each occurrence on right can cancel one occurrence on left
  - $\{1, 1, 2, 3\} - \{1, 2, 3, 4\} = \{1\}$
- Union, intersection no longer distribute
  - $\{1\} \cap (\{1\} \cup \{1\})$ vs. $(\{1\} \cap \{1\}) \cup (\{1\} \cap \{1\})$
  - $\{1\} \cap \{1, 1\}$ vs. $\{1\} \cup \{1\}$
  - $\{1\}$ $\neq$ $\{1,1\}$

Bag-projection (π) and duplicates

Make	Model	Color
Toyota	Prius	Gray
Toyota	Prius	Red
Honda	Accord	Green
Honda	Accord	Red
Honda	Accord	Red
Ford	Echo	Red
Ford	Echo	Gray
Ford	Echo	White

- Consider a relation R modeling cars for sale
- Bag-projection (π) does not eliminate duplicate tuples (as in set-projection)
  - What does $\pi_{Make}(R)$ return?
  - How to eliminate duplicates?

Duplicates important for summaries ("how many")

Duplicate elimination (δ)

Make	Model	Color
Toyota	Prius	Gray
Toyota	Prius	Red
Honda	Accord	Green
Honda	Accord	Red
Honda	Accord	Red
Ford	Echo	Red
Ford	Echo	Gray
Ford	Echo	White

- Consider a relation R modeling cars for sale
- δ turns a bag into a set
- $\delta(\pi_{Make}(R))$ is a set

Make
Honda
Ford
Toyota

Summarizing groups of tuples (1)

Student	Year	Dept	Course	Grade
Xiao	2009	CS	A08	B-
Xiao	2009	CS	A48	B
Xiao	2009	CS	A65	B+
Xiao	2009	Math	A23	B
Xiao	2009	Math	A30	B+
Xiao	2009	Math	A37	A
Xiao	2010	CS	B07	B
Xiao	2010	CS	B09	B-
Xiao	2010	CS	B36	B-
Xiao	2010	CS	B58	B
Xiao	2010	Math	B24	A-
Xiao	2010	Math	B41	B
Xiao	2010	Stats	B52	B-
Xiao	2011	CS	C24	B+
Xiao	2011	CS	C43	A-
Xiao	2011	CS	C69	A

All courses Xiao has taken

All courses Xiao took in 2010

All math courses Xiao took in 2010

Summarizing groups of tuples (2)

Student	Year	Dept	Course	Grade
Xiao	2009	CS	A08	B-
Xiao	2009	CS	A48	B
Xiao	2009	CS	A65	B+
Xiao	2009	Math	A23	B
Xiao	2009	Math	A30	B+
Xiao	2009	Math	A37	A
Xiao	2010	CS	B07	B
Xiao	2010	CS	B09	B-
Xiao	2010	CS	B36	B-
Xiao	2010	CS	B58	B
Xiao	2010	Math	B24	A-
Xiao	2010	Math	B41	B
Xiao	2010	Stats	B52	B-
Xiao	2011	CS	C24	B+
Xiao	2011	CS	C43	A-
Xiao	2011	CS	C69	A

How to summarize this??

Student	Dept	Year	Course	Grade
Xiao	CS	2009	?	?
Xiao	Math	2009	?	?
Xiao	CS	2010	?	?
Xiao	Math	2010	?	?
Xiao	Stats	2010	B52	B-
Xiao	CS	2011	?	?

These columns are easy… equal for every tuple in a group

Show the best grade? Worst grade? Average?

Summarizing groups of tuples (3)

- Description #1: want to output a single tuple which summarizes a set of related tuples
- Description #2: want to "collapse" a set of tuples into a single, "representative" tuple
- Questions
  - How to identify related tuples (set to collapse)?

    => Grouping key: a subset of attributes to test for equality
  - How to collapse a column into a value (summarize it)?

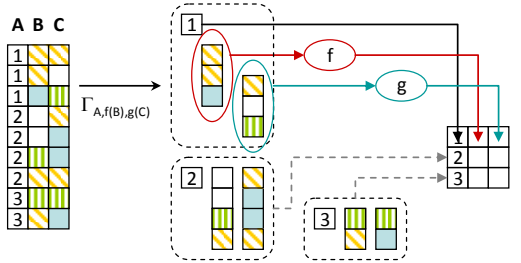
    => Use an aggregation function (sum, count, avg, min, max, …)

Grouping (Γ)

- Duplicates useful when computing statistics
  - min, max, sum, count, average, …
- $\Gamma_{A,B,C,f(X),g(Y),h(Z)}(R)$ computes aggregate values using some attributes as a grouping key
  - Implicit projection (drops unreferenced attributes)
  - A, B, C is the *grouping key*
  - X, Y, Z are *attributes* to aggregate
  - *f*, *g*, *h* are *aggregating* functions to apply
- Aggregating function should be commutative
  - f(x,y) = f(y,x)

Grouping (Γ)

- All tuples having the same key go to same group
  - One output tuple for each unique key
  - Output "group total" for each non-key attribute in group



Example instance:
A is Employee level (1: Employee, 2: Manager, 3: Executive)
B is Age, so **f(B)** can represent average Age
C is Salary, so **g(C)** can represent average Salary

Duplicates and grouping

Make	Model	Color
Toyota	Prius	Gray
Toyota	Prius	Red
Honda	Accord	Green
Honda	Accord	Red
Honda	Civic	Red
Ford	Echo	Red
Ford	Echo	Gray
Ford	Echo	White

- Consider a relation R modeling cars for sale
- $\Gamma_{Make,count(*)}(R)$ returns?
  - The number of cars of each make

Make	Count
Toyota	2
Honda	3
Ford	3

Duplicates important for summaries ("how many")

Sorting (τ)

- $\tau_L(R)$ sorts tuples in R on list of attributes L
  - If L is A1, A2, …,An tuples sorted first by A1. Ties are broken based on A2;…; Ties that remain after An broken arbitrarily.
  - Default: ascending order; With '-' in front: descending order
- Example: $\tau_{\text{-Count, Make}}(R)$

Make	Count
Toyota	2
Honda	3
Ford	3

$\xrightarrow{\tau}$

Make	Count
Ford	3
Honda	3
Toyota	2

Descending count

Alphabetical order when count is equal

- Sorted relations not in the R.A. value domain!

  => τ must be root operator of query tree**

The dangling tuple problem

- Consider the following query
  - $\tau_{\text{-Total}}(\rho_{\text{Name,Total}}(\Gamma_{\text{Name,sum(Value)}}(\text{Emp} \bowtie \text{Sales})))$
  - "List employees and their total sales in descending order"

Emp

EID	Name
1	Mary
2	Xiao
3	Jaspreet

⋈

Sales

EID	Value	…
1	20	…
3	10	…
3	15	…

=

Name	Total
Jaspreet	25
Mary	20

Xiao?

- *Join hides fact that Xiao has no sales!*
  - *Challenge: rewrite query to include Xiao's zero*

*What's **not** there can be very important*

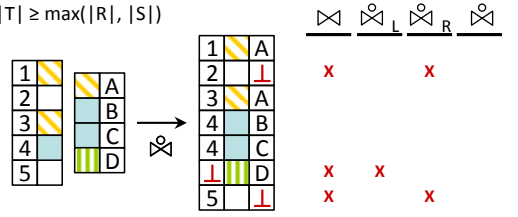
Extending the "inner" join

- All joins so far resemble intersection

  => Tuples with no match discarded ("dangling")
- Sometimes desirable to output dangling tuples
  - List all employees and their sales total (even if zero)
- Problem: what to use as missing half of tuple?
  - Introduce special value ⊥ (null)
  - Pad dangling tuples as needed to match schema
  - Note: ⊥ technically outside R.A. value domain

Outer join (⟗)

- T = R⟗ S computes the "outer" join of R (left) and S (right)
  - Like normal join, but all tuples from R and S appear in output
  - Pad (left, right, or all) dangling tuples with ⊥ or NULL
    - **LEFT** — Tuples in inner join padded with tuples in R that have no matching tuples in S.
    - **RIGHT** — Tuples in inner join padded with tuples in S that have no matching tuples in R.
    - **FULL** — Tuples in inner join padded with tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R.
  - *Natural, equi-, and theta- variants still apply*
  - |T| ≥ max(|R|, |S|)



Outer join (⟕) examples

r_1

Employee	Department
Smith	sales
Black	production
White	production

r_2

Department	Head
production	Mori
purchasing	Brown

$r_1 \bowtie_{LEFT} r_2$

Employee	Department	Head
Smith	sales	NULL
Black	production	Mori
White	production	Mori

$r_1 \bowtie_{RIGHT} r_2$

Employee	Department	Head
Black	production	Mori
White	production	Mori
NULL	purchasing	Brown

$r_1 \bowtie_{FULL} r_2$

Employee	Department	Head
Smith	Sales	NULL
Black	production	Mori
White	production	Mori
NULL	purchasing	Brown

Outer join in action

- Consider the following query
  - $\tau_{\text{-Total}}(\rho_{\text{Name,Total}}(\Gamma_{\text{Name,sum(Value)}}(\text{Emp} ⟕ \text{Sales})))$
  - "List employees and their total sales in descending order"

Emp

EID	Name
1	Mary
2	Xiao
3	Jaspreet

⟕

Sales

EID	Value	…
1	20	…
3	10	…
3	15	…

=

Name	Total
Jaspreet	25
Mary	20
Xiao	⊥

Extended projection

- $\pi_{x=E}(R)$ computes column x from expression E
  - Arithmetic (z=3*x + y)
  - String manipulation (substring, capitalization)
  - Some conditional expressions
- Example:

$\tau_{\text{-Total}}(\pi_{\text{Name,Total=T?T:0}}(\rho_{\text{Name,T}}(\Gamma_{\text{Name,sum(Value)}}(\text{Emp} ⟕ \text{Sales}))))$

Emp

EID	Name
1	Mary
2	Xiao
3	Jaspreet

⟕

Sales

EID	Value	…
1	20	…
3	10	…
3	15	…

=

Name	Total
Jaspreet	25
Mary	20
Xiao	0

Coming next…

- SQL (Structured Query Algebra)