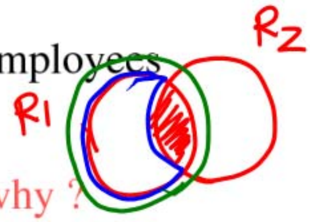
Derived RA Operations

1) Intersection

2) Most importantly: Join

Set Operations: Intersection

- Difference: all tuples both in R1 and in R2
- Notation: R1∩ R2
- R1, R2 must have the same schema
- R1∩R2 has the same schema as R1, R2
- Example
  - UnionizedEmployees ∩ RetiredEmployees
- Intersection is derived:
  - $\underline{R1 \cap R2} = \underline{R1} - (R1 - R2)$ why ?



Joins

$\times + ?$

- Theta join
- Natural join
- Equi-join
- Semi-join
- Inner join
- Outer join
- etc.

Theta Join

- A join that involves a predicate
- Notation: $R1 \bowtie_{\theta} R2$ where θ is a condition
- Input schemas: $R1(A1,\ldots,An)$, $R2(B1,\ldots,Bm)$
- $\{A1,\ldots An\} \cap \{B1,\ldots,Bm\} = \phi$
- Output schema: $S(A1,\ldots,An,B1,\ldots,Bm)$
- Derived operator:

$$R1 \bowtie_{\theta} R2 = \sigma_{\theta}(R1 \, x \, R2)$$

Theta-Join

- R3 := R1 JOIN_C R2
  - Take the product R1 * R2.
  - Then apply SELECT_C to the result.
- As for SELECT, *C* can be any boolean-valued condition.
  - Historic versions of this operator allowed only A theta B, where theta was =, <, etc.; hence the name "theta-join."

Example

Sells(

bar,	beer,	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

)

Bars(

name,	addr
Joe's	Maple St.
Sue's	River Rd.

)

BarInfo := Sells JOIN $_{\text{Sells.bar = Bars.name}}$ Bars

BarInfo(

bar,	beer,	price,	name,	addr
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Joe's	Maple St.
Sue's	Bud	2.50	Sue's	River Rd.
Sue's	Coors	3.00	Sue's	River Rd.

)

Natural Join

- Notation: $R1 \bowtie R2$
- Input Schema: $R1(A1, \ldots, An)$, $R2(B1, \ldots, Bm)$
- Output Schema: $S(C1,\ldots,Cp)$
  - Where $\{C1, \ldots, Cp\} = \{A1, \ldots, An\} \cup \{B1, \ldots, Bm\}$
- Meaning: combine all pairs of tuples in R1 and R2 that agree on the attributes:
  - $\{A1,\ldots,An\} \cap \{B1,\ldots, Bm\}$ (called the join attributes)
- Equivalent to a cross product followed by selection
- Example **Employee** $\bowtie$ **Dependents**

Natural Join Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependents

SSN	Dname
999999999	Emily
777777777	Joe

Employee $\bowtie$ Dependents =

$\Pi_{\text{Name, SSN, Dname}}(\sigma_{\text{SSN=SSN2}}(\text{Employee} \times \rho_{\text{SSN2, Dname}}(\text{Dependents}))$

Name	SSN	Dname
John	999999999	Emily
Tony	777777777	Joe

Natural Join

- R=

A	B
X	Y
X	Z
Y	Z
Z	V

S=

B	C
Z	U
V	W
Z	V

- $R \bowtie S$ =

A	B	C
X	Z	U
X	Z	V
Y	Z	U
Y	Z	V
Z	V	W

Natural Join

- Given the schemas R(A, B, C, D), S(A, C, E), what is the schema of $R \bowtie S$?

- Given R(A, B, C), S(D, E), what is $R \bowtie S$?

- Given R(A, B), S(A, B), what is $R \bowtie S$?

Equi-join

- Most frequently used in practice:

$$R1 \bowtie_{A=B} R2$$

- Natural join is a particular case of equi-join
- A lot of research on how to do it efficiently

Summary of Relational Algebra

- Basic primitives:

E ::= R
| $\sigma_C(E)$
| $\pi_{A1, A2, ..., An}(E)$
| E1 X E2
| E1 $\cup$ E2
| E1 - E2
| $\rho_{S(A1, A2, \ldots, An)}(E)$

- ~~Abbreviations~~: Derivated

| E1 JOIN E2
| E1 JOIN_{C} E2
| E1 $\cap$ E2

Relational Algebra

- Six basic operators, many derived
- Combine operators in order to construct queries: relational algebra expressions, usually shown as trees

Behind the Scene: Other query languages?

Find phone and name of bookstores that supply "Gone with the Wind":

Procedural

Books(title, BStorID) ⋈ Bookstore(BStorID, phone, name)

Relational algebra:

- Join Books and Bookstores over the BookstoreID.
- Restrict to tuples for the book "Gone with the Wind":.
- Project over StoreName and StorePhone.

Declarative

Relational calculus:

- Get StoreName and StorePhone such that there exists a title BK with the same BookstoreID value and ~~with a~~ BookTitle value of "Gone with the wind".

So what's the difference?

Building Complex Expressions

- Algebras allow us to express sequences of operations in a natural way.
- Example
  - in arithmetic algebra: $(x + 4)*(y - 3)$
  - in stack "algebra": T.push(S.pop())
- Relational algebra allows the same.
- Three notations, just as in arithmetic:
  1. Sequences of assignment statements.
  2. Expressions with several operators.
  3. Expression trees.

Sequences of Assignments

- Create temporary relation names.
- Renaming can be implied by giving relations a list of attributes.
- Example: R3 := R1 JOIN_C R2 can be written:

  R4 := R1 * R2

  R3 := SELECT_C (R4)

Expressions with Several Operators

- Example: the theta-join R3 := R1 JOIN$_C$ R2 can be written: R3 := SELECT$_C$ (R1 * R2)
- Precedence of relational operators:
  1. Unary operators --- select, project, rename --- have highest precedence, bind first.
  2. Then come products and joins.
  3. Then intersection.
  4. Finally, union and set difference bind last.
- But you can always insert parentheses to force the order you desire.

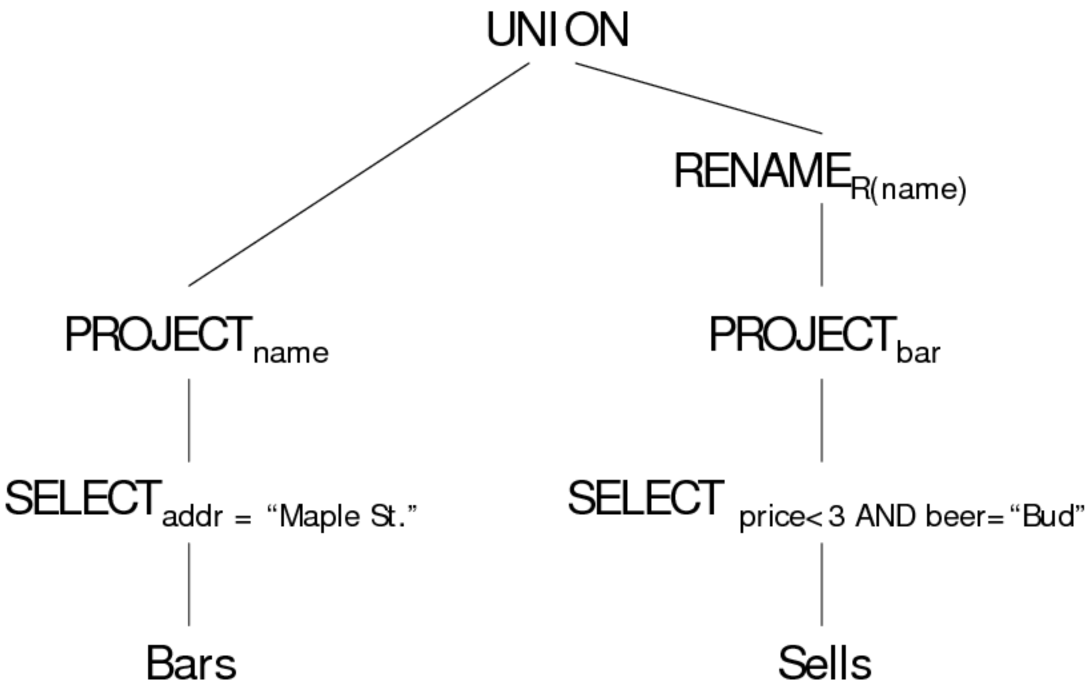
Expression Trees

- Leaves are operands --- either variables standing for relations or particular, constant relations.
- Interior nodes are operators, applied to their child or children.

Example

- Given Bars(name, addr), Sells(bar, beer, price), find the names of all the bars that are either on Maple St. or sell Bud for less than $3.

As a Tree:


UNION
RENAME R(name)
PROJECT name
PROJECT bar
SELECT addr = “Maple St.”
SELECT price< 3 AND beer=“Bud”
Bars
Sells

Q: How to do this?

- Using Sells(bar, beer, price), find the bars that sell two different beers at the same price.

More Queries

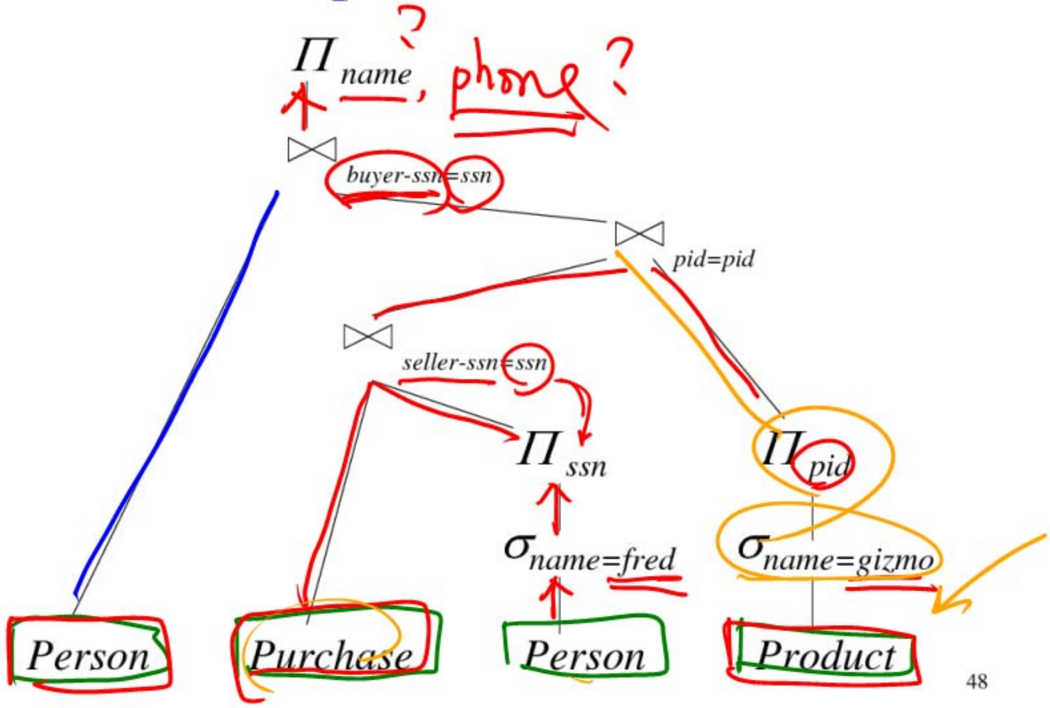
Product (pid, name, price, category, maker-cid)
Purchase (buyer-ssn, seller-ssn, store, pid)
Company (cid, name, stock price, country)
Person(ssn, name, phone number, city)

Note:
•in Purchase: buyer-ssn, seller-ssn are **foreign keys** in Person, pid is **foreign key** in Product;
•in Product maker-cid is a **foreign key** in Company

Find phone numbers of people who bought gizmos from Fred.

Find telephony products that somebody bought

Expression Tree



Exercises

Product (pid, name, price, category, maker-cid)
Purchase (buyer-ssn, seller-ssn, store, pid)
Company (cid, name, stock price, country)
Person(ssn, name, phone number, city)

Ex #1: Find people who bought telephony products.
Ex #2: Find names of people who bought American products
Ex #3: Find names of people who bought American products and did not buy French products
Ex #4: Find names of people who bought American products and they live in Champaign.
Ex #5: Find people who bought stuff from Joe or bought products from a company whose stock prices is more than $50.

Operations on Bags ↔ Set
(and why we care)

- Union: {a,b,b,c} U {a,b,b,b,e,f,f} = {a,a,b,b,b,b,b,c,e,f,f}
  - *add* the number of occurrences
- Difference: {a,b,b,b,c,c} – {b,c,c,c,d} = {a,b,b}
  - subtract the number of occurrences
- Intersection: {a,b,b,b,c,c}   {b,b,c,c,c,c,d} = {b,b,c,c}
  - minimum of the two numbers of occurrences
- Selection: preserve the number of occurrences
- Projection: preserve the number of occurrences (no duplicate elimination)
- Cartesian product, join: no duplicate elimination

Read the book for more detail

Summary of Relational Algebra

- Why bother ? Can write any RA expression directly in C++/Java, seems easy.
- Two reasons:
  – Each operator admits sophisticated implementations (think of $\bowtie$, σ_C)
  – Expressions in relational algebra can be rewritten: optimized

Glimpse Ahead: Efficient Implementations of Operators

- $\sigma_{(\textbf{age} >= 30 \text{ AND } \textbf{age} <= 35)}(\textbf{Employees})$
  - Method 1: scan the file, test each employee
  - Method 2: use an index on **age**
  - Which one is better ? Depends a lot…
- **Employees** $\bowtie$ **Relatives**
  - Iterate over Employees, then over Relatives
  - Iterate over Relatives, then over Employees
  - Sort Employees, Relatives, do “merge-join”
  - “hash-join”
  - etc

Glimpse Ahead: Optimizations

Product (<u>pid</u>, name, price, category, maker-cid)
Purchase (buyer-ssn, seller-ssn, store, pid)
Person(<u>ssn</u>, name, phone number, city)

- Which is better:

  $\sigma_{price>100}(\text{Product}) \bowtie (\text{Purchase} \bowtie \sigma_{city=sea}\text{Person})$

  $(\sigma_{price>100}(\text{Product}) \bowtie \text{Purchase}) \bowtie \sigma_{city=sea}\text{Person}$

- Depends ! This is the optimizer's job…