Relational Algebra

INF 551
Wensheng Wu

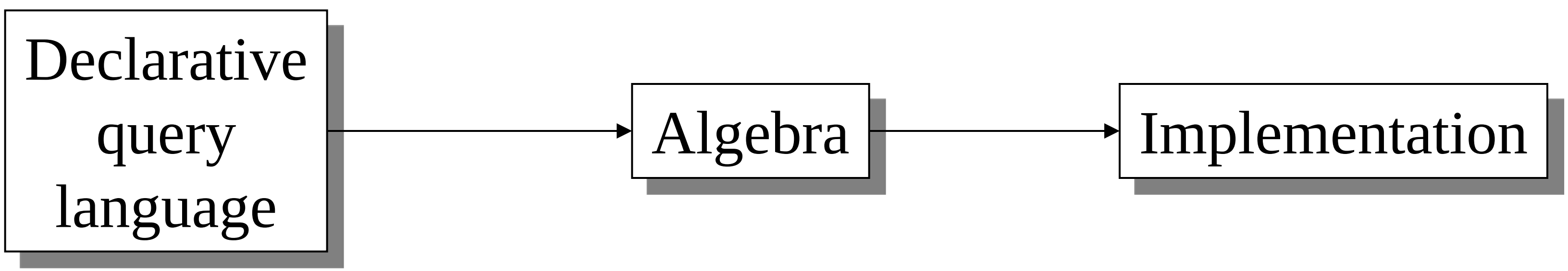
Querying the Database

- Goal: specify what we want from our database

  Find all the employees who earn more than $50,000 and pay taxes in Los Angeles County.
- Could write in Java/Python, but bad idea
- Instead use *high-level query languages:*
  - Practical: SQL
  - Theoretical: Relational Algebra, Datalog

Relational Algebra

- Formalism for creating new relations from existing ones
- Its place in the big picture:



SQL
Relational calculus

Relational algebra
Relational bag algebra

Procedural

Motivation: The Stack

- To use the "stack" data structure in my program, I need to know
  - what a stack looks like
  - what (useful) operations I can perform on a stack
    - PUSH and POP
- Next, I look for an implementation of stack
  - browse the Web
  - find many of them
  - choose one, say LEDA (Library for Efficient Data types and Algorithms, in C++)

Motivation: The Stack (cont.)

- LEDA already implement PUSH and POP
- It also gives me a simple language L, in which to define a stack and call PUSH and POP
  - S = init_stack(int);
  - S.push(3); S.push(5);
  - int x = S.pop();
- Can also define an expression of operations on stacks
  - T = init_stack(int);
  - T.push(S.pop());

Motivation: The Stack (cont.)

- To summarize, I know
  - definition of stack
  - its operations (PUSH, POP): that is, a stack algebra
  - an implementation called LEDA, which tells me how to call PUSH and POP in a language L
  - I can use these implementations to manipulate stacks
  - LEDA hides the implementation details
  - LEDA optimizes implementation of PUSH and POP

Now Contrast It with Rel. Databases

- To summarize, I know
  - definition of stack
  - its operations (PUSH, POP): that is, a stack algebra
  - an implementation called LEDA, which tells me how to call PUSH and POP in a language L
  - I can use these implementations to manipulate stacks
  - LEDA hides the implementation details
  - LEDA optimizes implementation of PUSH and POP

def of relations

relational algebra (RA)

RA expression

operation and query optimization

Outline

- Motivation
- Relational algebra
- Relational bag algebra
- Extended RA

What is an “Algebra”

- Mathematical system consisting of:
  - *Operands* --- variables or values from which new values can be constructed.
  - *Operators* --- symbols denoting procedures that construct new values from given values.

What is Relational Algebra?

- An algebra whose operands are relations or variables that represent relations.
- Operators are designed to do the most common things that we need to do with relations in a database.
  - The result is an algebra that can be used as a *query language* for relations.

Relational Algebra at a Glance

- Operators: relations as input, new relation as output
- Five basic RA operations:
  - Basic Set Operations
    - union, difference
  - Selection: σ
  - Projection: π
  - Cartesian Product: X (sometimes denoted as *)
- When our relations have conflicting attribute names:
  - Renaming: ρ
- Derived operations:
  - Intersection
  - Joins (theta join, equi-join, natural, semi-join, etc.)

Five Basic RA Operations

Set Operations

- Union, difference
- Both are binary operations

Set Operations: Union

- Union: all tuples in R1 or R2
- Notation: R1 U R2
- R1, R2 must have the same schema
- R1 U R2 has the same schema as R1, R2
- Example:
  - ActiveEmployees U RetiredEmployees

Set Operations: Difference

- Difference: all tuples in R1 and not in R2
- Notation: R1 – R2
- R1, R2 must have the same schema
- R1 - R2 has the same schema as R1, R2
- Example
  - AllEmployees - RetiredEmployees

Selection

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- Unary operation
- *c* is a condition: boolean expression built from =, <, >, and, or, not
- Output schema: same as input schema
- Find all employees with salary more than $40,000:
  - $\sigma_{Salary > 40000}$(Employee)

Selection Example

Employee

SSN	Name	DepartmentID	Salary
999999999	John	1	30,000
777777777	Tony	1	32,000
888888888	Alice	2	45,000

Find all employees with salary more than $40,000.

$\sigma_{\mathit{Salary} > 40000}(\text{Employee})$

SSN	Name	DepartmentID	Salary
888888888	Alice	2	45,000

Ullman: Selection

- R1 := SELECT$_C$ (R2)
  - C is a condition (as in “if” statements) that refers to attributes of R2.
  - R1 is all those tuples of R2 that satisfy C.

Example

Relation Sells:

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

JoeMenu := SELECT$_{bar=\text{“Joe's”}}$(Sells):

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75

Projection

- Unary operation: returns certain columns
- It eliminates duplicate tuples! (so set semantics)
- Notation: $\Pi_{A1,\ldots,An}(R)$
- Input schema $R(B1,\ldots,Bm)$
- Condition: $\{A1, \ldots, An\} \subseteq \{B1, \ldots, Bm\}$
- Output schema $S(A1,\ldots,An)$
- Example: project social-security number and names:
  - $\Pi_{SSN,\ Name}(Employee)$

Projection Example

Employee

SSN	Name	DepartmentID	Salary
999999999	John	1	30,000
777777777	Tony	1	32,000
888888888	Alice	2	45,000

$\mathbf{\Pi_{SSN,\ Name}\ (Employee)}$

SSN	Name
999999999	John
777777777	Tony
888888888	Alice

Projection

- R1 := PROJ$_L$ (R2)
  - *L* is a list of attributes from the schema of R2.
  - R1 is constructed by looking at each tuple of R2, extracting the attributes on list *L*, in the order specified, and creating from those components a tuple for R1.
  - Eliminate duplicate tuples, if any.

Example

Relation Sells:

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

Prices := $PROJ_{beer,price}$(Sells):

beer	price
Bud	2.50
Miller	2.75
Miller	3.00

Duplicates removed!

Cartesian Product

- Each tuple in *R1* paired with each tuple in *R2*
- Notation: *R1 x R2*
- Input schemas *R1(A1,…,An)*, *R2(B1,…,Bm)*
- Condition: $\{A1,\ldots,An\} \cap \{B1,\ldots Bm\} = \Phi$
- Output schema is *S(A1, …, An, B1, …, Bm)*
- Example: **Employee x Dependents**
- Very rare in practice; but joins are very common

Cartesian Product Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependents

EmployeeSSN	Dname
999999999	Emily
777777777	Joe

Employee x Dependents

Name	SSN	EmployeeSSN	Dname
John	999999999	999999999	Emily
John	999999999	777777777	Joe
Tony	777777777	999999999	Emily
Tony	777777777	777777777	Joe

Product

- R3 := R1 * R2
  - Pair each tuple t1 of R1 with each tuple t2 of R2.
  - Concatenation t1t2 is a tuple of R3.
  - Schema of R3 is the attributes of R1 and R2, in order.
  - But beware attribute *A* of the same name in R1 and R2: use R1.*A* and R2.*A*.

Example: R3 := R1 * R2

R1(

A,	B
1	2
3	4

)

R2(

B,	C
5	6
7	8
9	10

)

R3(

A,	R1.B,	R2.B,	C
1	2	5	6
1	2	7	8
1	2	9	10
3	4	5	6
3	4	7	8
3	4	9	10

)

Renaming

- Does not change the relational instance
- Changes the relational schema only
- Notation: $\rho_{S(B1,\ldots,Bn)}(R)$
  - *If without S, assume output relation is also named R or you do not care about its name*
- Input schema: $R(A1, \ldots, An)$
- Output schema: $S(B1, \ldots, Bn)$
- Example:

$$\rho_{LastName,\ SocSocNo}(Employee)$$

Renaming Example

Employee

Name	SSN
John	999999999
Tony	777777777

$$\rho_{LastName,\ SocSocNo}\ \mathbf{(Employee)}$$

Employee

LastName	SocSocNo
John	999999999
Tony	777777777

Renaming

- The RENAME operator gives a new schema to a relation.
- R1 := $\text{RENAME}_{\text{R1(A1,…,A}n\text{)}}$(R2) makes R1 be a relation with attributes A1,…,A*n* and the same tuples as R2.
- Simplified notation: R1(A1,…,A*n*) := R2.

Example

Bars(

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

)

R(bar, addr) := Bars

R(

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

)

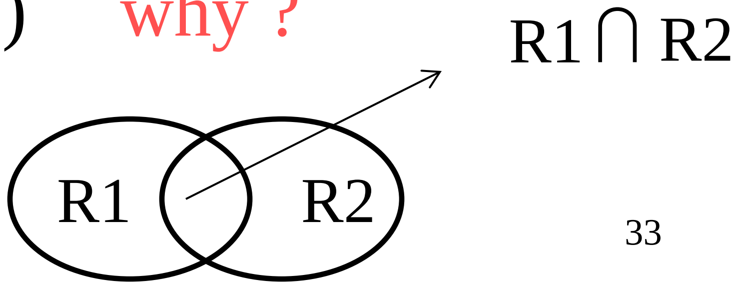
Derived RA Operations

1) Intersection

2) Most importantly: Join

Set Operations: Intersection

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



Joins

- 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:

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

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 + projection
- Example **Employee** $\bowtie$ **Dependents**

Natural Join Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependents

SSN	Dname
999999999	Emily
777777777	Joe

Employee $\bowtie$ Dependents =

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

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 $R \bowtie S$?

Natural Join

- A frequent type of join connects two relations by:
  - Equating attributes of the same name, and
  - Projecting out one copy of each pair of equated attributes.
- Called *natural* join.
- Denoted R3 := R1 JOIN R2.

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(

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

)

BarInfo := Sells JOIN Bars

Note Bars.name has become Bars.bar to make the natural join "work."

BarInfo(

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

)

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

Semijoin

- $R \ltimes S = \Pi_{A1,\ldots,An}(R \bowtie S)$
  - *Tuples in R that pair with some tuple in S on common attributes*

- Where the schemas are:
  - Input: *R(A1,….An), S(B1,…,Bm)*
  - Output: *T(A1,…,An)*

- R(X,Y), S(Y,Z): $R \ltimes S = R \bowtie (\Pi_Y S)$

Example

- 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

$\Pi_{A1,\ldots,An}(R \bowtie S)$

A	B
X	Z
Y	Z
Z	V

Note: duplicate (x,z) (y,z) removed

Example

- R=

A	B
X	Y
X	Z
Y	Z
Z	V

S=

B	C
Z	U
V	W
Z	V

$\Pi_B S$

B
Z
V

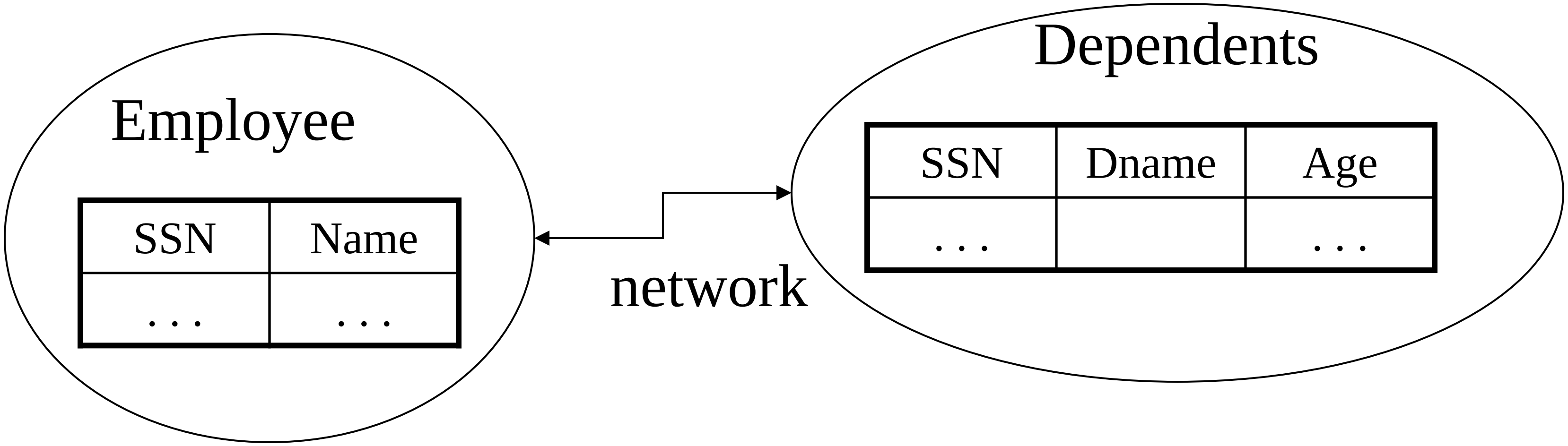
Note: duplicate z removed

$R \bowtie (\Pi_B S)$

A	B
X	Z
Y	Z
Z	V

Semijoins in Distributed Databases

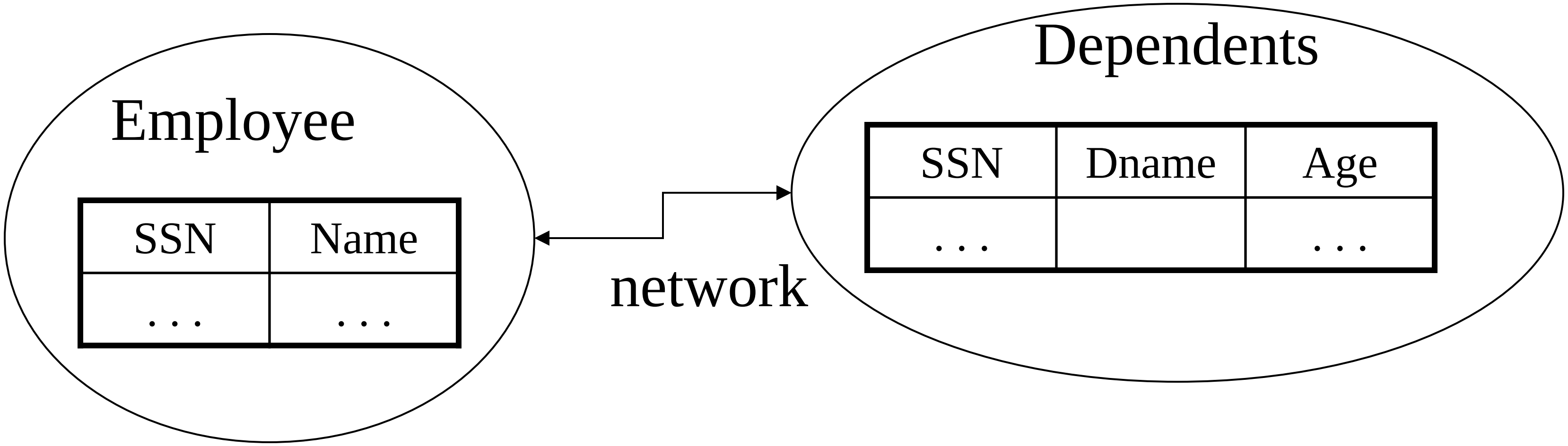
- Semijoins are used in distributed databases



$$\text{Employee} \bowtie_{ssn=ssn} (\sigma_{\ age>71} (\text{Dependents}))$$

Semijoins in Distributed Databases

- Semijoins are used in distributed databases



$$\text{Employee} \bowtie_{ssn=ssn} (\sigma_{\,age>71} (\text{Dependents}))$$

$$T = \Pi_{SSN}\, \sigma_{\,age>71} (\text{Dependents})$$

$$R = \text{Employee} \bowtie T$$

$$\text{Answer} = R \bowtie \text{Dependents}$$

Relational Algebra

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

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

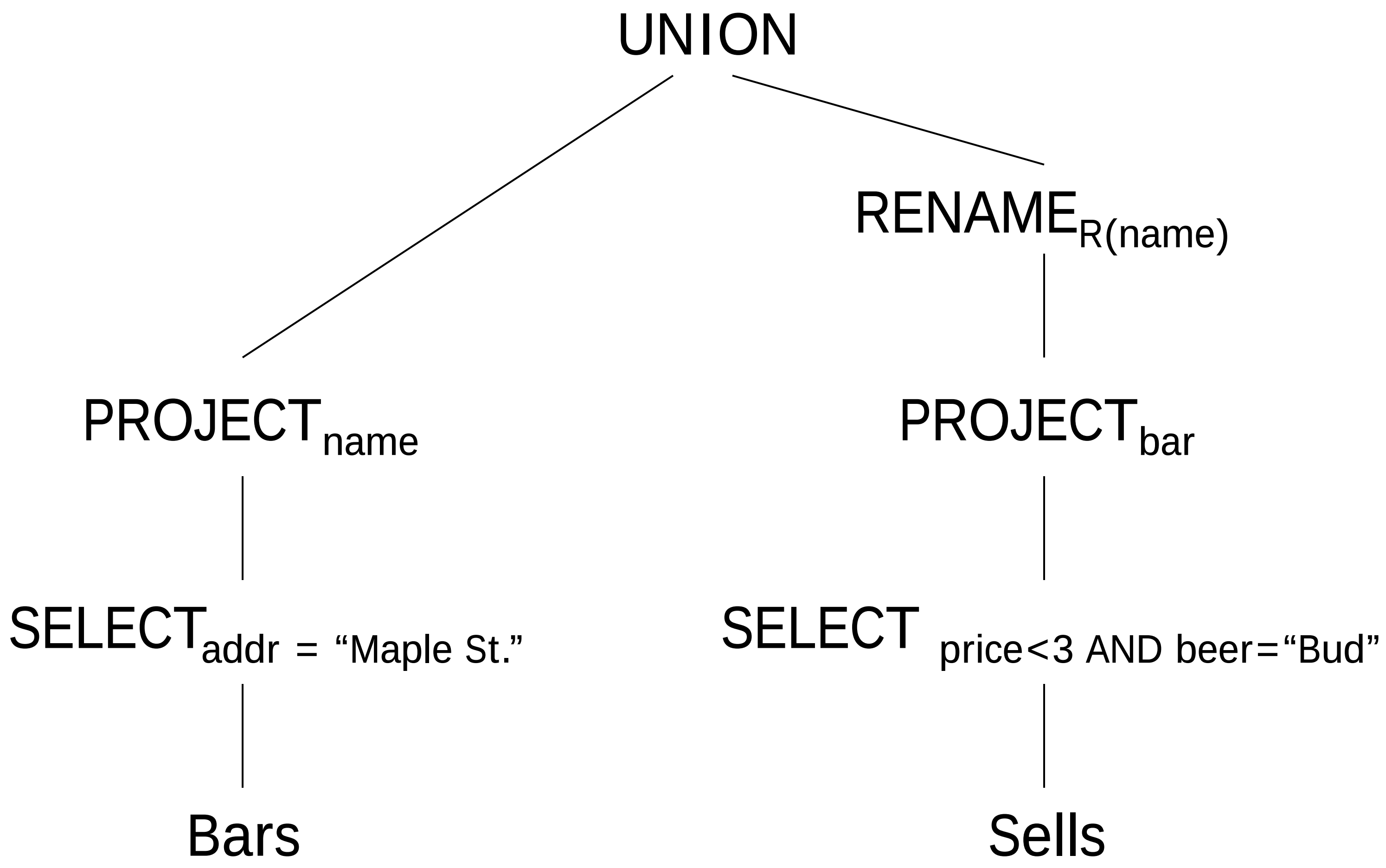
- Leaves are operands --- either variables standing for relations or particular, constant relations.
- Interior nodes are operators, applied to their child or children.

Example

- Using the relations Bars(name, addr) and 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:

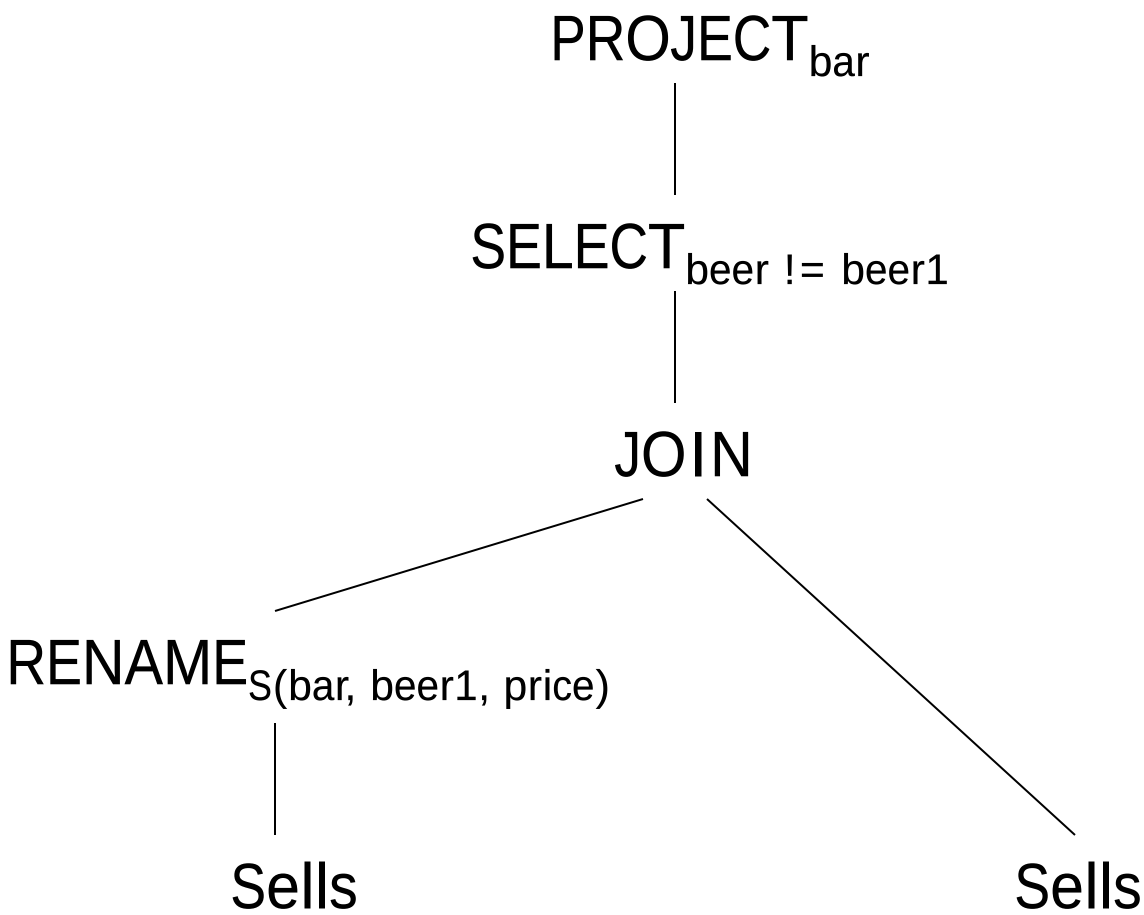
- Using the relations Bars(name, addr) and Sells(bar, beer, price), find the names of all the bars that are either on Maple St. or sell Bud for less than $3.



Example

- Using Sells(bar, beer, price), find the bars that sell two different beers at the same price.
- Strategy: by renaming, define a copy of Sells, called S(bar, beer1, price). The natural join of Sells and S consists of quadruples (bar, beer, beer1, price) such that the bar sells both beers at this price.

The Tree



PROJECT bar
SELECT beer != beer1
JOIN
RENAME S(bar, beer1, price)
Sells
Sells

Schemas for Interior Nodes

- An expression tree defines a schema for the relation associated with each interior node.
- Similarly, a sequence of assignments defines a schema for each relation on the left of the := sign.

Schema-Defining Rules

- For union, intersection, and difference, the schemas of the two operands must be the same, so use that schema for the result.
- Selection: schema of the result is the same as the schema of the operand.
- Projection: list of attributes tells us the schema.

Schema-Defining Rules

- Product: the schema is the attributes of both relations.
  - Use R.*A*, etc., to distinguish two attributes named *A*.
- Theta-join: same as product.
- Natural join: use attributes of both relations.
  - Shared attribute names are merged.
- Renaming: the operator tells the schema.

Complex Queries

Product (<u>pid</u>, name, price, category, maker_cid)
Purchase (buyer_ssn, seller_ssn, store, pid)
Company (<u>cid</u>, name, stock price, country)
Person (<u>ssn</u>, name, phone number, city)

Note:
- maker_cid in Product refers to cid in Company
- buyer_ssn and seller_ssn in Purchase refers to ssn in Person
- pid in Purchase refers to pid in Product

Query:

Find names of people who bought gizmos from Fred.

SQL

- Select buyer.name

  From person buyer, person seller,

  purchase, product

  Where product.name = 'Gizmo' and

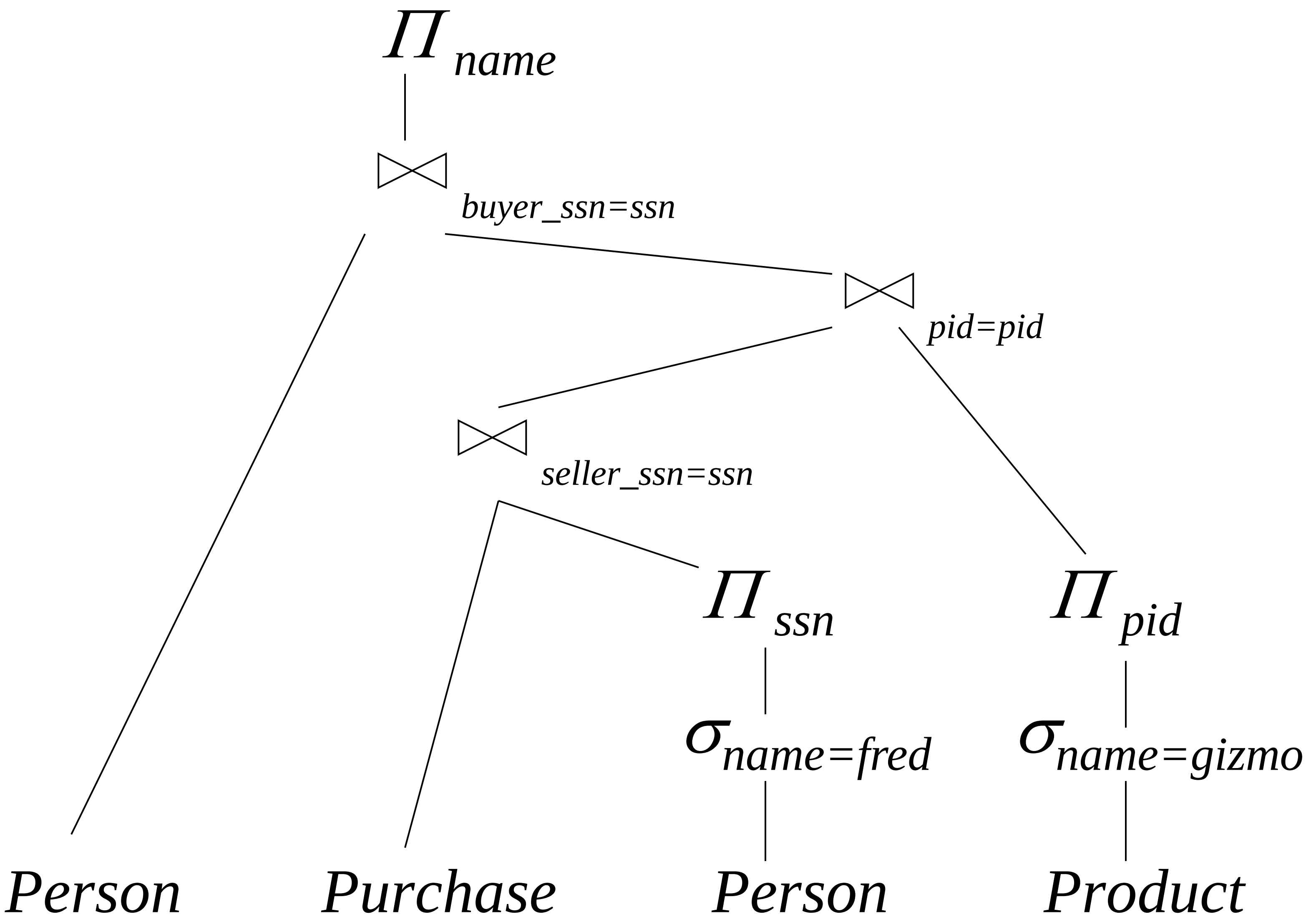
  purchase.pid = product.pid and

  purchase.buyer_ssn = buyer.ssn and

  purchase.seller_ssn = seller.ssn and

  seller.name = 'Fred'

Expression Tree



Exercises

Product (<u>pid</u>, name, price, category, maker_cid)
Purchase (buyer_ssn, seller_ssn, store, pid)
Company (<u>cid</u>, name, stock price, country)
Person(<u>ssn</u>, name, phone number, city)

Ex #1: Find people who bought telephony products.

Ex #2: Find names of people who bought American products

Exercises

Product (<u>pid</u>, name, price, category, maker_cid)
Purchase (buyer_ssn, seller_ssn, store, pid)
Company (<u>cid</u>, name, stock price, country)
Person(<u>ssn</u>, name, phone number, city)

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 live in Los Angeles.

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 #5:

Find names of people who bought stuff from Joe or bought products from a company whose stock price is more than $50.

Relational Bag Algebra

Relational Algebra on Bags

- A *bag* is like a set, but an element may appear more than once.
  - *Multiset* is another name for "bag."
- Example: {1,2,1,3} is a bag. {1,2,3} is also a bag that happens to be a set.
- Bags also resemble lists, but order in a bag is unimportant.
  - Example: {1,2,1} = {1,1,2} as bags, but [1,2,1] != [1,1,2] as lists.

Why Bags?

- SQL, the most important query language for relational databases is actually a bag language.
  - SQL will eliminate duplicates, but usually only if you ask it to do so explicitly.
- Some operations, like projection, are much more efficient on bags than sets.

Operations on Bags

- Selection applies to each tuple, so its effect on bags is like its effect on sets.
- Projection also applies to each tuple, but as a bag operator, we do not eliminate duplicates.
- Products and joins are done on each pair of tuples, so duplicates in bags have no effect on how we operate.

Example: Bag Selection

R(

A,	B
1	2
5	6
1	2

)

S(

B,	C
3	4
7	8

)

$\text{SELECT}_{A+B<5}$ (R) =

A	B
1	2
1	2

Example: Bag Projection

R(

A,	B
1	2
5	6
1	2

)

S(

B,	C
3	4
7	8

)

PROJECT$_A$ (R) =

A
1
5
1

Example: Bag Product

R(

A,	B
1	2
5	6
1	2

)

S(

B,	C
3	4
7	8

)

R * S =

A	R.B	S.B	C
1	2	3	4
1	2	7	8
5	6	3	4
5	6	7	8
1	2	3	4
1	2	7	8

Example: Bag Theta-Join

R(

A,	B
1	2
5	6
1	2

)

S(

B,	C
3	4
7	8

)

R JOIN $_{R.B<S.B}$ S =

A	R.B	S.B	C
1	2	3	4
1	2	7	8
5	6	7	8
1	2	3	4
1	2	7	8

Bag Union

- Union, intersection, and difference need new definitions for bags.
- An element appears in the union of two bags the sum of the number of times it appears in each bag.
- Example: {1,2,1} UNION {1,1,2,3,1} = {1,1,1,1,1,2,2,3}

Bag Intersection

- An element appears in the intersection of two bags the minimum of the number of times it appears in either.
- Example: {1,2,1} INTER {1,2,3} = {1,2}.

Bag Difference

- An element appears in the difference $A - B$ of bags as many times as it appears in *A*, minus the number of times it appears in *B*.
  - But never less than 0 time.
- Example: {1,2,1} – {1,2,2,3} = {1}.

Beware: Bag Laws != Set Laws

- Not all algebraic laws that hold for sets also hold for bags.
- For one example, the commutative law for union (R UNION S = S UNION R) *does* hold for bags.
  - Since addition is commutative, adding the number of times x appears in R and S doesn't depend on the order of R and S.

An Example of Inequivalence

- Set union is *idempotent*, meaning that S UNION S = S.
- However, for bags, if x appears n times in S, then it appears $2n$ times in S UNION S.
- Thus S UNION S != S in general.

Set operation in SQL

- Union, intersect, except are implemented in PostgreSQL
  - Follow set-semantics
  - Remove duplicates

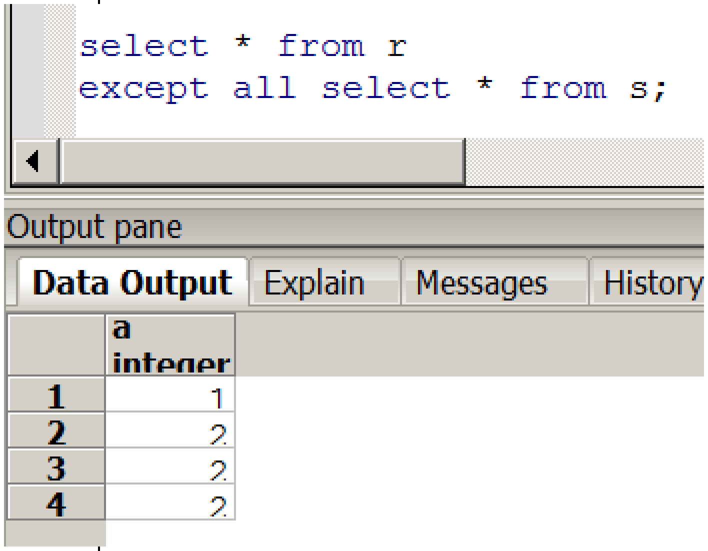
- Union all, intersect all, and except
  - Follow bag-semantics

Example (postgresql)

- create table r (a int);
- insert into r values(1);
- insert into r values(2);
- insert into r values(2);
- insert into r values(2);
- insert into r values(2);
- insert into r values(2);

- create table s (a int);
- insert into s values(2);
- insert into s values(2);

- select * from r except all select * from s;



Extended RA

The Extended Algebra

1. DELTA = eliminate duplicates from bags.
2. TAU = sort tuples.
3. *Extended projection* : arithmetic, duplication of columns.
4. GAMMA = grouping and aggregation.
5. OUTERJOIN: adds “dangling tuples” = tuples that do not join with anything.

The Extended Algebra: Symbols

1. DELTA (δ)
2. TAU (τ)
3. *Extended projection* : arithmetic, duplication of columns.
4. GAMMA (γ)
5. OUTERJOIN (⟕, ⟖, ⟗)

Duplicate Elimination

- R1 := DELTA(R2).
- R1 consists of one copy of each tuple that appears in R2 one or more times.

Example: Duplicate Elimination

R =

A	B
1	2
3	4
1	2

DELTA(R) =

A	B
1	2
3	4

Sorting

- R1 := TAU_L (R2).
  - *L* is a list of some of the attributes of R2.
- R1 is the list of tuples of R2 sorted first on the value of the first attribute on *L*, then on the second attribute of *L*, and so on.
  - Break ties arbitrarily.
- TAU is the only operator whose result is neither a set nor a bag.

Example: Sorting

R =

A	B
1	2
3	4
5	2

TAU_B (R) = [(5,2), (1,2), (3,4)]

Extended Projection

- Using the same $PROJ_L$ operator, we allow the list L to contain arbitrary expressions involving attributes, for example:
  1. Arithmetic on attributes, e.g., $A+B$.
  2. Duplicate occurrences of the same attribute.

Example: Extended Projection

R =

A	B
1	2
3	4

$PROJ_{A+B,A,A}$ (R) =

A+B	A1	A2
3	1	1
7	3	3

Aggregation Operators

- Aggregation operators are typically used together with grouping operator.
- They apply to one column of a table and produce a single result.
- The most important examples: SUM, AVG, COUNT, MIN, and MAX.

Example: Aggregation

R =

A	B
1	3
3	4
3	2

SUM(A) = 7
COUNT(A) = 3
MAX(B) = 4
AVG(B) = 3

Grouping Operator

- R1 := GAMMA_L (R2). *L* is a list of elements that are either:
  1. Individual (*grouping*) attributes.
  2. AGG(*A*), where AGG is one of the aggregation operators and *A* is an attribute.

Applying $GAMMA_L$(R)

- Group *R* according to all the grouping attributes on list *L*.
  - That is, form one group for each distinct list of values for those attributes in *R*.
- Within each group, compute AGG(*A*) for each aggregation on list *L*.
- Result has grouping attributes and aggregations as attributes. One tuple for each list of values for the grouping attributes and their group's aggregations.

Example: Grouping/Aggregation

R =

A	B	C
1	2	3
4	5	6
1	2	5

GAMMA$_{A,B,\mathrm{AVG}(C)}$ (R) = ??

First, group *R* :

A	B	C
1	2	3
1	2	5
4	5	6

Then, average *C* within groups:

A	B	AVG(C)
1	2	4
4	5	6

Outerjoin

- Suppose we join R JOIN$_C$ S.
- A tuple of R that has no tuple of S with which it joins is said to be *dangling*.
  - Similarly for a tuple of S.
- Outerjoin preserves dangling tuples by padding them with a special NULL symbol in the result.

Example: Outerjoin

R =

A	B
1	2
4	5

S =

B	C
2	3
6	7

(1,2) joins with (2,3), but the other two tuples are dangling.

R OUTERJOIN S =

A	B	C
1	2	3
4	5	NULL
NULL	6	7

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

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

Optimizations

Product (pid, name, price, category, maker_cid)
Purchase (ssn, seller_ssn, store, pid) // ssn is buyer ssn
Person(ssn, name, phone number, city)

- Which is better:

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

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

- Depends ! This is the optimizer’s job…

Finally: RA has Limitations!

- Cannot compute “transitive closure”

Name1	Name2	Relationship
Fred	Mary	Father
Mary	Joe	Cousin
Mary	Bill	Spouse
Nancy	Lou	Sister

- Find all direct and indirect relatives of Fred
- Cannot express in RA, need recursion !!!
- But note that new SQL standard permits recursion.
  – With common table expression (not available in MySQL though)