CS411
Database Systems

04: Relational Algebra

Why Do We Learn This?

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 Champaign-Urbana.
- Could write in C++/Java, but bad idea
- Instead use *high-level query languages:*
  - Theoretical: Relational Algebra, Datalog
  - Practical: SQL

  - Relational algebra: a basic set of operations on relations that provide the *basic principles*.

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

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

SQL language

operation and query optimization

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.

Q: Example algebra?

- Arithmetic algebra. Linear algebra.
- What are operands?
- What are operators?

What is Relational Algebra?

- An algebra whose operands are relations or variables that represent relations.
- Operators are designed to do 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 (no intersection, no complement)
  - Selection: σ
  - Projection: π
  - Cartesian Product: X
- When our relations have attribute names:
  - Renaming: ρ
- Derived operations:
  - Intersection, complement
  - Joins (natural,equi-join, theta join, semi-join)

Basic RA Operations

Set Operations

- Union, difference
- 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: $\sigma_c(R)$
- *c* is a condition: =, <, >, 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.

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

SSN	Name	DepartmentID	Salary
888888888	Alice	2	45,000

Projection

- Unary operation: returns certain columns
- Eliminates duplicate tuples !
- 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

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

SSN	Name
999999999	John
777777777	Tony
888888888	Alice

Q: Comparing projection and selection?

- Think of relation as a table.
- How are they similar?
- How are they different?
- Why do you need both?

Cartesian Product

- Each tuple in *R1* 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)*
- Notation: *R1 x R2*
- 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

Renaming

- Does not change the relational instance
- Changes the relational schema only
- Notation: $\rho_{B1,\ldots,Bn}(R)$
- Input schema: *R(A1, …, An)*
- Output schema: *S(B1, …, Bn)*
- Example:

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

Renaming Example

Employee

Name	SSN
John	999999999
Tony	777777777

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

LastName	SocSocNo
John	999999999
Tony	777777777

Behind the Scene: Why Codd invented rel. algebra?

- Codd proposed R.A. right up front-- in the 1970 CACM paper on relational model.
- As a query language? No.
- For defining "data health" by derivability.

> In Section 2 operations on relations and two types of redundancy are defined and applied to the problem of maintaining the data in a consistent state. This is bound to become a serious practical problem as more and more different types of data are integrated together into common data banks.

> Suppose θ is a collection of operations on relations and each operation has the property that from its operands it yields a unique relation (thus natural join is eligible, but join is not). A relation R is *θ-derivable* from a set S of relations if there exists a sequence of operations from the collection θ which, for all time, yields R from members of S.

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:
  - R1 ∩ 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, …, An), R2(B1, …, Bm)*
- Output Schema: *S(C1,…,Cp)*
  - Where *{C1, …, Cp} = {A1, …, An} U {B1, …, Bm}*
- Meaning: combine all pairs of tuples in R1 and R2 that agree on the attributes:
  - *{A1,…,An}* $\cap$ *{B1,…, 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 =

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

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⋈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 ⋈ S ?
- Given R(A, B, C), S(D, E), what is R ⋈ S ?
- Given R(A, B), S(A, B), what is R ⋈ 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:

| 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":

Relational algebra:

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

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.

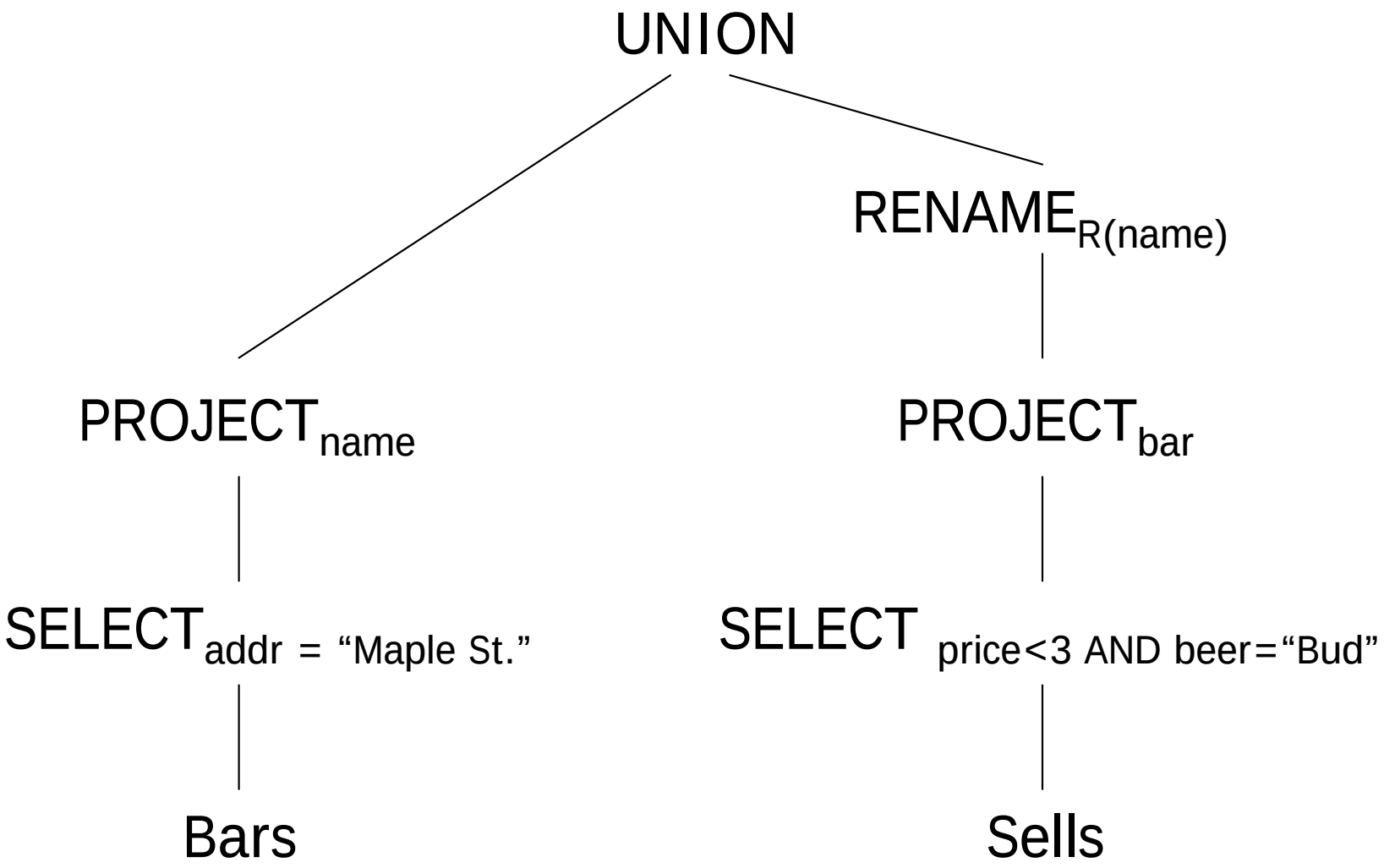
Expression Trees

- 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

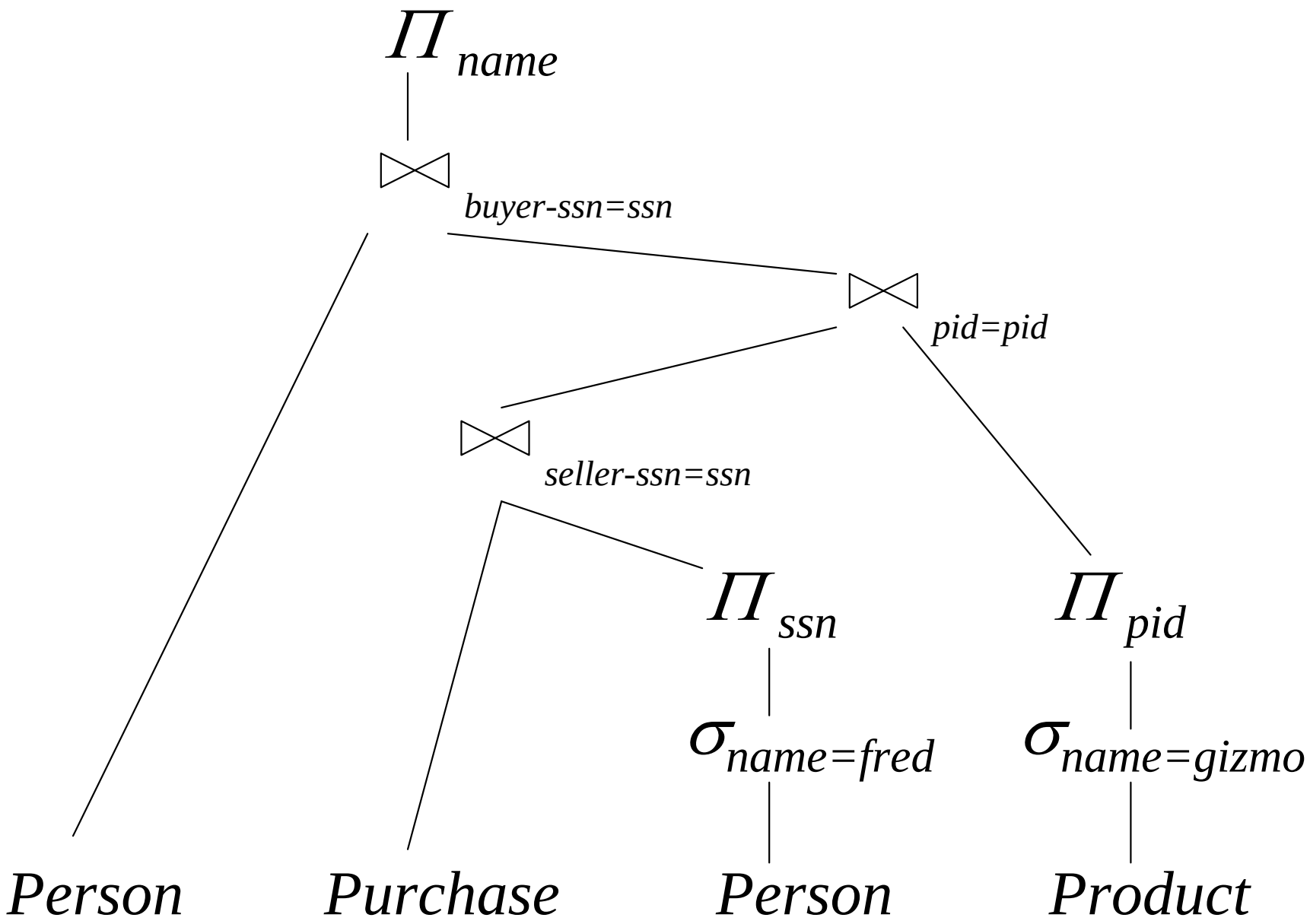
Product (pid, name, price, category, maker-cid)
Purchase (buyer-ssn, seller-ssn, store, pid)
Company (cid, name, stock price, country)
Person(ssn, name, phone number, city)

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

- Which is better:

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

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

- Depends ! This is the optimizer's job…