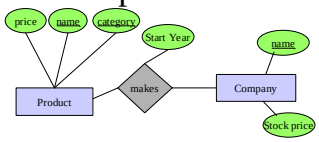
A Note: Many-many Relationships to Relations

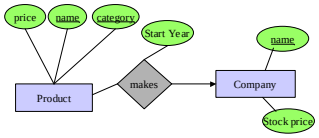


Product(price, name, category)

Makes(product-name,category,company-name,startyear)

Company(name,stockprice)

One-many Relationships to Relations



As before, or (better):

Product(price, name, category,company-name,startyear)

Company(name,stockprice)

Lecture 9: The Relational Algebra

Friday, October 13 2000

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 New Jersey.
- 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*.

Relational Algebra at a Glance

- Operators: relations as input, new relation as output
- Five basic RA operators:
  - Basic Set Operators
    - union, difference (no intersection, no complement)
  - Selection: σ
  - Projection: π
  - Cartesian Product: X
- Derived operators:
  - Intersection, complement
  - Joins (natural,equi-join, theta join, semi-join)
- When our relations have attribute names:
  - Renaming: ρ

Set Operations

- Binary operations
- Union, difference, intersection
  - Intersection can be expressed in other ways

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

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

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,…,Bm)*
- Condition: *{A1, …, An}* ⊆ *{B1, …, Bm}*
- Output schema *S(A1,…,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

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 often

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

Derived Operations

- Intersection is derived:
  - R1 ∩ R2 = R1 – (R1 – R2) why ?
  - There is another way to express it (later)
- Most importantly: joins, in many variants

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} ∩ {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 ?

Theta Join

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

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

Semijoin

- $R \ltimes S = \Pi_{A1,…,An}(R \bowtie S)$
- Where the schemas are:
  - Input: *R(A1,…An), S(B1,…,Bm)*
  - Output: *T(A1,…,An)*

Relational Algebra

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

Complex 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

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 Seattle.
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,d}
  - 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

Reading assignment: 4.6.2 - 4.6.6

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**
  - Many implementation methods (some researchers built careers out of that)

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…

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 to write C program