Lecture 3: Relational Algebra and SQL

Tuesday, January 9, 2001

Outline

- Relational Algebra: 4.2 (except 4.2.5)
- SQL: 5.2, 5.3, 5.4

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

- Operators: relations as input, new relation as output
- Five basic RA operators:
  - Set Operators
    - union, difference
    - Selection: σ
  - Projection: π
  - Cartesian Product: $\times$
- Derived operators:
  - Intersection, complement
  - Joins (natural,equi-join, theta join, semi-join)
- When our relations have attribute names:
  - Renaming: ρ

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: 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 > $40,000:
  - $\sigma_{\text{Salary} > 40000}(\text{Employee})$

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.

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

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:
  - $\Pi_{\text{SSN, Name}}$ (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,…,An} ∩ {B1,…Bm} = Φ
- 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)

13

Renaming Example

Employee

Name	SSN
John	999999999
Tony	777777777

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

LastName	SocSocNo
John	999999999
Tony	777777777

14

Derived Operations

- Intersection can be derived:
  - $R1 \cap R2 = R1 - (R1 - R2)$
  - There is another way to express it (later)
- Most importantly: joins, in many variants

15

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**

16

Natural Join Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependents

SSN	Dname
999999999	Emily
777777777	Joe

Employee $\bowtie$ **Dependents =**

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

Name	SSN	Dname
John	999999999	Emily
Tony	777777777	Joe

17

Another Natural Join 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

18

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 ?

19

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} = ϕ
- Output schema: S(A1,…,An,B1,…,Bm)
- Derived operator:
  R1 $\bowtie_\theta$ R2 = σ_θ (R1 x R2)

20

Semijoin

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

21

Summary of Relational Algebra

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

22

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

23

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}$\cap${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

24

SQL Introduction

- Standard language for querying and manipulating data

  Structured **Q**uery **L**anguage

- Many standards out there: SQL92, SQL2, SQL3, SQL99
- Vendors support various subsets of these, but all of what we'll be talking about.
- Works on bags, rather than sets

- Basic construct:

```
SELECT …
FROM …
WHERE …
```

25

Selections

Company(sticker, name, country, stockPrice)

Find all US companies whose stock is > 50:

```
SELECT *
FROM Company
WHERE country="USA" AND stockPrice > 50
```

Output schema: R(sticker, name, country, stockPrice)

26

Selections

What you can use in WHERE:

- attribute names of the relation(s) used in the FROM.
- comparison operators: =, <>, <, >, <=, >=
- apply arithmetic operations: stockprice*2
- operations on strings (e.g., "||" for concatenation).
- Lexicographic order on strings.
- Pattern matching: s LIKE p
- Special stuff for comparing dates and times.

27

The LIKE operator

- s LIKE p: pattern matching on strings
- p may contain two special symbols:
  - % = any sequence of characters
  - _ = any single character

Company(sticker, name, address, country, stockPrice)
Find all US companies whose address contains "Mountain":

```
SELECT *
FROM Company
WHERE country="USA" AND
      address LIKE "%Mountain%"
```

- Needed in the 1st assignment !

28

Projections

Select only a subset of the attributes

```
SELECT name, stockPrice
FROM Company
WHERE country="USA" AND stockPrice > 50
```

Input schema: Company(sticker, name, country, stockPrice)
Output schema: R(name, stock price)

29

Projections with Renamings

Rename the attributes in the resulting table

```
SELECT name AS company, stockprice AS price
FROM Company
WHERE country="USA" AND stockPrice > 50
```

Input schema: Company(sticker, name, country, stockPrice)
Output schema: R(company, price)

30

Eliminating Duplicates

```
SELECT  DISTINCT country
FROM    Company
WHERE   stockPrice > 50
```

Without DISTINCT the result is a bag

31

Ordering the Results

```
SELECT   name, stockPrice
FROM     Company
WHERE    country="USA" AND stockPrice > 50
ORDERBY  country, name
```

Ordering is ascending, unless you specify the DESC keyword.

Ties are broken by the second attribute on the ORDERBY list, etc.

32

Joins

Product (pname, price, category, maker)
Purchase (buyer, seller, store, product)
Company (cname, stockPrice, country)
Person(per-name, phoneNumber, city)

Find names of people living in Seattle that bought gizmo products, and the names of the stores they bought from

```
SELECT   per-name, store
FROM     Person, Purchase
WHERE    per-name=buyer AND city="Seattle"
                        AND product="gizmo"
```

33

Disambiguating Attributes

Find names of people buying telephony products:

```
SELECT   Person.name
FROM     Person, Purchase, Product
WHERE        Person.name=buyer
         AND  product=Product.name
         AND  Product.category="telephony"
```

Product (name, price, category, maker)
Purchase (buyer, seller, store, product)
Person(name, phoneNumber, city)

34

Tuple Variables

Find pairs of companies making products in the same category

```
SELECT  product1.maker, product2.maker
FROM      Product AS product1, Product AS product2
WHERE     product1.category=product2.category
      AND  product1.maker <> product2.maker
```

Product (name, price, category, maker)

35

Tuple Variables

•Tuple variables introduced automatically by the system:

Product (name, price, category, maker)

```
SELECT  name
FROM    Product
WHERE   price > 100
```

Becomes:

```
SELECT  Product.name
FROM    Product AS Product
WHERE   Product.price > 100
```

Doesn't work when Product occurs more than once.

36

Meaning (Semantics) of SQL Queries

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions

1. Nested loops:
        Answer = {}
        for x1 in R1 do
            for x2 in R2 do
                …..
                    for xn in Rn do
                        if Conditions
                            then Answer = Answer U {(a1,…,ak)}
         return Answer
```


Meaning (Semantics) of SQL Queries

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions

2. Parallel assignment

        Answer = {}
        for all assignments x1 in R1, …, xn in Rn do
            if Conditions then Answer = Answer U {(a1,…,ak)}
        return Answer
```

Doesn't impose any order !

Meaning (Semantics) of SQL Queries

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions
```

3. Translation to Relational algebra:

$\Pi_{a1,,…,ak}$ (σ $_{Conditions}$ (R1 x R2 x … x Rn))

Select-From-Where queries are precisely Select-Project-Join

First Unintuitive SQLism

```
SELECT   R.A
FROM   R, S, T
WHERE  R.A=S.A    OR   R.A=T.A
```

Looking for R ∩ (S U T)

But what happens if T is empty?

Union, Intersection, Difference

```
(SELECT  name
 FROM     Person
 WHERE    City="Seattle")
    UNION
(SELECT  name
 FROM     Person, Purchase
 WHERE   buyer=name AND store="The Bon")
```

Similarly, you can use INTERSECT and EXCEPT.

You must have the same attribute names (otherwise: rename).

Conserving Duplicates

The UNION, INTERSECTION and EXCEPT operators operate as sets, not bags.

```
(SELECT  name
 FROM     Person
 WHERE    City="Seattle")
    UNION  ALL
(SELECT  name
 FROM     Person, Purchase
 WHERE   buyer=name AND store="The Bon")
```

Subqueries

A subquery producing a single tuple:

```
SELECT Purchase.product
FROM   Purchase
WHERE  buyer =
          (SELECT  name
           FROM    Person
           WHERE   ssn = "123456789");
```

In this case, the subquery returns one value.

If it returns more, it's a run-time error.

43

Can say the same thing without a subquery:

```
SELECT Purchase.product
FROM   Purchase, Person
WHERE  buyer = name AND ssn = "123456789"
```

Is this query equivalent to the previous one ?

44

Subqueries Returning Relations

Find companies who manufacture products bought by Joe Blow.

```
SELECT  Company.name
FROM    Company, Product
WHERE   Company.name=maker
        AND  Product.name  IN
                (SELECT product
                 FROM   Purchase
                 WHERE  buyer = "Joe Blow");
```

Here the subquery returns a set of values

45

Subqueries Returning Relations

Equivalent to:

```
SELECT  Company.name
FROM    Company, Product, Purchase
WHERE   Company.name=maker
        AND  Product.name  = product
        AND  buyer = "Joe Blow"
```

Is this query equivalent to the previous one ?

46

Subqueries Returning Relations

You can also use: s > ALL R
s > ANY R
EXISTS R

Product (pname, price, category, maker)
Find products that are more expensive than all those produced
By "Gizmo-Works"

```
SELECT  name
FROM    Product
WHERE  price >  ALL (SELECT price
                     FROM    Purchase
                     WHERE  maker="Gizmo-Works")
```

47

Question for Database Fans

- Can we express this query as a single SELECT-FROM-WHERE query, without subqueries ?
- Hint: show that all SFW queries are monotone (figure out what this means). A query with **ALL** is not monotone

48

Conditions on Tuples

```
SELECT  Company.name
FROM    Company, Product
WHERE   Company.name=maker
        AND  (Product.name,price)  IN
             (SELECT product, price)
              FROM   Purchase
              WHERE  buyer = "Joe Blow");
```


Correlated Queries

Movie (<u>title, year</u>, director, length)
Find movies whose title appears more than once.

```
SELECT  title
FROM    Movie AS x
WHERE   year < ANY
              (SELECT  year
               FROM    Movie
               WHERE   title =  x.title);
```

correlation

Note (1) scope of variables (2) this can still be expressed as single SFW

Complex Correlated Query

Product (pname, price, category, maker, year)

- Find products (and their manufacturers) that are more expensive than all products made by the same manufacturer before 1972

```
SELECT  pname, maker
FROM    Product AS x
WHERE   price > ALL  (SELECT  price
                      FROM    Product AS y
                      WHERE   x.maker = y.maker AND y.year < 1972);
```

Powerful, but much harder to optimize !

Exercises: write RA and SQL expressions

Product (pname, price, category, maker)
Purchase (buyer, seller, store, product)
Company (cname, stock price, country)
Person(per-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.