CSE 544
Principles of Database Management Systems

Magdalena Balazinska

Fall 2006

Lecture 3 - Relational Model

References

- E.F. Codd. A relational model of data for large shared data banks. Communications of the ACM, 1970. Sections 1.1-1.4 and all of Section 2.
- R&G book, chapters 3.6, 4, and 5

Outline

- **Relational model**
  - Discussion of Codd's paper Sections 1, 2.2, and 2.3 (no slides)
- **Relational algebra as introduced by Codd**
  - Discussion of Codd's paper Section 2 (no slides)
- **Modern relational algebra and calculus**
- **Brief review of SQL**
- **Logical data independence with views (next lecture)**

Relational Queries

- **Query inputs and outputs are relations**
- Query evaluation
  - Input: instances of input relations
  - Output: instance of output relation

Relational Algebra

- **Query language** associated with relational model
- **Queries specified in an operational manner**
  - A query gives a step-by-step procedure
- **Relational operators**
  - Take one or two relation instances as argument
  - Return one relation instance as result
  - Easy to **compose** into **relational algebra expressions**

Relational Operators

- Selection: $\sigma_{condition}(S)$
  - Condition is Boolean combination ($\wedge, \vee$) of terms
  - Term is: attr. op constant, attr. op attr.
  - Op is: <, <=, =, $\neq$, >=, or >
- Projection: $\pi_{list\text{-}of\text{-}attributes}(S)$
- Union ($\cup$), Intersection ($\cap$), Set difference (–),
- Cross-product or cartesian product ($\times$)
- Join: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
- Division: R/S, Rename $\rho(R(F),E)$

Selection & Projection Examples

Patient

no	name	zip	disease
1	p1	98125	flu
2	p2	98125	heart
3	p3	98120	lung
4	p4	98120	heart

$\pi_{zip,disease}$(Patient)

zip	disease
98125	flu
98125	heart
98120	lung
98120	heart

$\sigma_{disease='heart'}$(Patient)

no	name	zip	disease
2	p2	98125	heart
4	p4	98120	heart

π_{zip} ($\sigma_{disease='heart'}$(Patient))

zip
98120
98125

Relational Operators

- Selection: $\sigma_{condition}(S)$
  - Condition is Boolean combination ($\wedge$, $\vee$) of terms
  - Term is: attr. op constant, attr. op attr.
  - Op is: $<$, $<=$, $=$, $\neq$, $>=$, or $>$
- Projection: $\pi_{list\text{-}of\text{-}attributes}(S)$
- Union ($\cup$), Intersection ($\cap$), Set difference ($-$),
- Cross-product or cartesian product ($\times$)
- Join: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
- Division: R/S, Rename $\rho(R(F),E)$

Cross-Product Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

P x V

P.age	P.zip	disease	name	V.age	V.zip
54	98125	heart	p1	54	98125
54	98125	heart	p2	20	98120
20	98120	flu	p1	54	98125
20	98120	flu	p2	20	98120

Different Types of Join

- **Theta-join**: $R \bowtie_\theta S = \sigma_\theta(R \times S)$
  - Join of R and S with a join condition θ
  - Cross-product followed by selection θ

- **Equijoin**: $R \bowtie_\theta S = \pi_A(\sigma_\theta(R \times S))$
  - Join condition θ consists only of equalities
  - Projection π_A drops all redundant attributes

- **Natural join**: $R \bowtie S = \pi_A(\sigma_\theta(R \times S))$
  - Equijoin
  - Equality on **all** fields with same name in R and in S

Theta-Join Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

$$P \bowtie_{P.age=V.age \wedge P.zip=A.zip \wedge P.age < 50} V$$

P.age	P.zip	disease	name	V.age	V.zip
20	98120	flu	p2	20	98120

Equijoin Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

$P \bowtie_{P.age=V.age} V$

age	P.zip	disease	name	V.zip
54	98125	heart	p1	98125
20	98120	flu	p2	98120

Natural Join Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

$P \bowtie V$

age	zip	disease	name
54	98125	heart	p1
20	98120	flu	p2

More Joins

- **Outer join**
  - Include tuples with no matches in the output
  - Use NULL values for missing attributes
- Variants
  - Left outer join
  - Right outer join
  - Full outer join

Outer Join Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu
33	98120	lung

Voters V

name	age	zip
p1	54	98125
p2	20	98120

$P \overset{o}{\bowtie} V$

age	zip	disease	name
54	98125	heart	p1
20	98120	flu	p2
33	98120	lung	null

Example of Algebra Queries

Q1: Names of patients who have heart disease

$\pi_{name}(\text{Voter} \bowtie (\sigma_{disease='heart'}(\text{AnonPatient}))$

More Examples

Using relations from Lecture 2

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,qty,price)
```

Q2: Name of supplier of parts with size greater than 10

$\pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10}\ (\text{Part}))$

Q3: Name of supplier of red parts or parts with size greater than 10

$\pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10}\ (\text{Part}) \cup \sigma_{pcolor='red'}\ (\text{Part})\ )\ )$

(Many more examples in the book)

Extended Operators of Relational Algebra

- Duplicate elimination (δ)
  - Since commercial DBMSs operate on multisets not sets
- Aggregate operators (γ)
  - Min, max, sum, average, count
- Grouping operators (γ)
  - Partitions tuples of a relation into “groups”
  - Aggregates can then be applied to groups
- Sort operator (τ)

Relational Calculus

- Alternative to relational algebra

- Declarative query language
  - Describe what we want NOT how to get it

- Tuple relational calculus query
  - **{ T | p(T) }**
  - Where T is a tuple variable
  - p(T) denotes a formula that describes T
  - Result: set of all tuples for which p(T) is true
  - Language for p(T) is subset of **first-order logic**

Sample TRC Query

Q1: Names of patients who have heart disease

$\{ T \mid \exists P \in \text{AnonPatient } \exists V \in \text{Voter}$

$(P.zip = V.zip \wedge P.age = V.age \wedge P.disease = \text{'heart'} \wedge T.name = V.name) \}$

Outline

- **Relational model**
  – Discussion of Codd's paper Sections 1, 2.2, and 2.3 (no slides)
- **Relational algebra as introduced by Codd**
  – Discussion of Codd's paper Section 2 (no slides)
- **Modern relational algebra and calculus**
- **Brief review of SQL: Incomplete!**
- **Logical data independence with views**

Structured Query Language: SQL

- Influenced by relational calculus
- Declarative query language
- Multiple aspects of the language
  - Data definition language
    - Statements to create, modify tables and views
  - Data manipulation language
    - Statements to issue queries, insert, delete data
  - More

SQL Query

Basic form: (plus many many more bells and whistles)

```
SELECT  <attributes>
FROM    <one or more relations>
WHERE   <conditions>
```

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT   *
FROM     Product
WHERE    category='Gadgets'
```

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

"selection"

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  PName, Price, Manufacturer
FROM    Product
WHERE   Price > 100
```

PName	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

"selection" and "projection"

Details

- Case insensitive:
  - Same: SELECT Select select
  - Same: Product   product
  - Different: ‘Seattle’ ‘seattle’

- Constants:
  - ‘abc’ - yes
  - “abc” - no

Eliminating Duplicates

```
SELECT   DISTINCT category
FROM     Product
```

→

Category
Gadgets
Photography
Household

Compare to:

```
SELECT   category
FROM     Product
```

→

Category
Gadgets
Gadgets
Photography
Household

Ordering the Results

```
SELECT   pname, price, manufacturer
FROM     Product
WHERE    category='gizmo' AND price > 50
ORDER BY price, pname
```

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

Ordering is ascending, unless you specify the DESC keyword.

Joins

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all products under $200 manufactured in Japan;
return their names and prices.

Join between Product and Company

```
SELECT   PName, Price
FROM     Product, Company
WHERE    Manufacturer=CName AND Country='Japan'
         AND Price <= 200
```

Tuple Variables

Person(pname, address, worksfor)
Company(cname, address)

```
SELECT   DISTINCT pname, address
FROM     Person, Company
WHERE    worksfor = cname
```

Which address ?

⇒

```
SELECT   DISTINCT Person.pname, Company.address
FROM     Person, Company
WHERE    Person.worksfor = Company.cname
```

⇒

```
SELECT   DISTINCT x.pname, y.address
FROM     Person AS x, Company AS y
WHERE    x.worksfor = y.cname
```

Subqueries Returning Relations

Company(<u>name</u>, city)
Product(<u>pname</u>, maker)
Purchase(<u>id</u>, product, buyer)

Return cities where one can find companies that manufacture products bought by Joe Blow

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

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')
```

Correlated Queries

Movie (title, year, director, length)

Find movies whose title appears more than once.

correlation

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

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

Aggregation

```
SELECT  avg(price)
FROM    Product
WHERE   maker="Toyota"
```

```
SELECT  count(*)
FROM    Product
WHERE   year > 1995
```

SQL supports several aggregation operations:

sum, count, min, max, avg

Except count, all aggregations apply to a single attribute

Grouping and Aggregation

```
SELECT   S
FROM     R1,…,Rn
WHERE    C1
GROUP BY a1,…,ak
HAVING   C2
```

Conceptual evaluation steps:

1. Evaluate FROM-WHERE, apply condition C1
2. Group by the attributes $a_1,\ldots,a_k$
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result

Read more about it in the book...