Topic 7:

Relational Algebra

Background

- Introduced by Codd (along with the Tuple Relational Calculus)
- Relational Algebra . . . :
  - Is procedural, like most programming languages

    — we need to supply an ordering of operations
  - Would not be a good replacement for SQL in a DBMS
  - Is a good introduction to the operators provided by SQL

Relational Operators (1 / 2)

Relations are *closed* under Relational Algebra operators

- That is, they accept relations as operands, and produce relations as results.
- Example: Integers are closed under $+$ and $-$.

The eight basic Relational Algebra operators are:

$\times$	$\bowtie$
$-$	π
$\div$	σ
$\cap$	$\cup$

Relational Operators (2 / 2)

The eight Relational Algebra operators can be grouped in two ways:

1. Set vs. Relational:
   - Set: $\cup$, $\cap$, –, $\times$, $\div$
   - Relational: σ, π, $\bowtie$
2. Fundamental vs. Derived:
   - Fundamental: σ, π, $\times$, $\cup$, –
   - Derived: $\cap$, $\bowtie$, $\div$

The Fundamental Operators

– Select (σ)

– Project (π)

– Cartesian Product ($\times$)

– Union ($\cup$)

– Difference ($-$)

Select (σ, sigma) (1 / 2)

- A unary (single argument) operator
- Chooses full tuples from a relation based on a condition
- Form:

Example(s):

List all information of the employees in department #5:

Who are the active suppliers in Paris?

Select (σ, sigma) (2 / 2)

Notes:

- Conditions may be as complex as is necessary
- Select is commutative:

$$\sigma_A(\sigma_B(\text{r})) \equiv \sigma_B(\sigma_A(\text{r}))$$

- Cascades of selects $\equiv$ conjunction in a single select:

$$\sigma_A(\sigma_B(\sigma_C(\text{r}))) \equiv \sigma_{A \wedge B \wedge C}(\text{r})$$

Project (π, pi) (1 / 2)

Pronunciation: PRO-ject (not pro-JECT, not PRAH-ject)

- Also a unary operator
- Chooses named columns from a relation
  - Resulting group of tuples may include duplicates . . .
  - . . . which we drop to preserve entity integrity
- Form:

Project (π, pi) (2 / 2)

Example(s):

Find the names & salaries of the employees in department 5:

Alternatively:

Cartesian Product ($\times$) (1 / 2)

- A binary operator (form: R $\times$ S)
- 'Marries' all pairings of tuples from the given relations
  - resulting cardinality = card(R) $\cdot$ card(S)
  - resulting degree = degree(R) + degree(S)

Example(s):

A

m	n
2	i
3	iv
7	x

B

o	p
3	β
7	α

Cartesian Product ($\times$) (2 / 2)

Example(s):

What are the names of the active suppliers of nuts?

The complete query:

$$\pi_{\text{Sname}}(\sigma_{\text{Status}>0 \wedge \text{Pname='Nut'}}(\sigma_{\text{S.S\# = SP.S\#}}(\sigma_{\text{SP.P\# = P.P\#}}(S \times (SP \times P)))))$$

For a visualization of this query step–by–step with sample data, see the handout:

"Examples of the Relation Algebra Operations σ, π, and $\times$"

Union ($\cup$) (1 / 2)

- Another binary operator (form: R $\cup$ S)
- Result contains all tuples of both relations w/o duplicates

Example(s):

"Create a table of the Phoenix and Tucson employee data."

Union (∪) (2 / 2)

To perform R ∪ S, R and S must be union compatible.

Definition: Union Compatible

. . . .

. . . .

Example(s):

Difference (−) (1 / 2)

- Do you remember this one from basic sets?

$$\{a, b, c, d, e, f\} - \{b, d, f, h\} = \{a, c, e\}$$

- Yet another binary operator (form: R − S)
- Result is a relation of tuples from R that are not also in S
- Note that R and S must be union compatible

Difference (−) (2 / 2)

Example(s):

X

s	t
a	12
m	4
e	6

Y

u	v
e	6
a	16
f	4

The Derived Operators

– Intersection ($\cap$)

– Join ($\bowtie$)

– Division ($\div$)

Intersection (∩) (1 / 2)

- YABO — Yet Another Binary Operator! (form: R ∩ S)
- Resulting relation has the tuples that appear in both operand relations
- As with difference, R and S must be union compatible

Intersection (∩) (2 / 2)

Example(s):

X

s	t
a	12
m	4
e	6

Y

u	v
e	6
a	16
f	4

X ∩ Y

s	t
e	6

Join (⋈) (1 / 3)

- YABO! (form: R $\bowtie_{\text{condition}}$ S)
- Join is used to exploit PK–FK connections

  (using it with other attributes is unwise!)

Join (⋈) (2 / 3)

Example(s):

What are the names of the parts that can be supplied by individual suppliers in quantity $>$ 200?

Without ⋈: $\pi_{\text{pname}}(\sigma_{\text{qty}>200}(\sigma_{\text{SP.P\#}=\text{P.P\#}}(\text{SP} \times \text{P})))$

With ⋈:

Join (⋈) (3 / 3)

Three join variations:

1. Theta Join: r $\bowtie_\theta$ s

2. Equijoin: r $\bowtie_\theta$ s

3. Natural Join: r $\bowtie$ s $\equiv \pi_{R \cup S}(r \bowtie_{r.a1=s.a1 \wedge r.a2=s.a2 \wedge \ldots} s)$

   where R and S are the attribute sets of r and s, respectively

Division (÷) (1 / 9)

Let α and β be relations, where $A - B$ is the set difference of the attributes of α & β

Definition of Relational Division:

Division ($\div$) (2 / 9)

Let's review multiplication and division of integers:

Ex: $4 * 6 = 24$, so $24/6 = 4$ and $24/4 = 6$.

Now, consider Cartesian Product and Division with relations:

M

c
4
8

N

d
3
1
7

$M \times N = O$

c	d
4	3
4	1
4	7
8	3
8	1
8	7

Division ($\div$) (3 / 9)

What purpose does Division serve?

Example(s):

Recall:

S

S#	Sname	Status	City

P

P#	Pname	Color	Weight	City

SP

S#	P#	Qty

Division (÷) (4 / 9) $\alpha \div \beta = \pi_{\text{A-B}}(\alpha) - \pi_{\text{A-B}}((\pi_{\text{A-B}}(\alpha) \times \beta) - \alpha)$

Example(s): (continued)

(Find the S#s of the suppliers that supply all parts of weight = 17)

"Find the X . . . " $\leftarrow$ X is S# (the matches we want)

". . . matched w/ all Y" $\leftarrow$ Y is the set of weight 17 P#s

Next, create the dividend (α) and divisor (β) relations:

Division (÷) (5 / 9)

Example(s): (continued)

The content of α and β:

α

S#	P#
S1	P1
S2	P3
S2	P5
S3	P3
S3	P4
S4	P6
S5	P1
S5	P2
S5	P3
S5	P4
S5	P5
S5	P6

β

P#
P2
P3

Do any suppliers supply ALL of β's parts?

Division (÷) (6 / 9)

Let's examine the definition in detail:

$$\alpha \div \beta = \pi_{\text{S\#}}(\alpha) - \pi_{\text{S\#}}((\pi_{\text{S\#}}(\alpha) \times \beta) - \alpha)$$

Division (÷) (7 / 9)

Looking at the data should help, too:

$\pi_{\text{S\#}}(\alpha) \times \beta$

S#	P#
S1	P2
S1	P3
S2	P2
S2	P3
S3	P2
S3	P3
S4	P2
S4	P3
S5	P2
S5	P3

α

S#	P#
S1	P1
S2	P3
S2	P5
S3	P3
S3	P4
S4	P6
S5	P1
S5	P2
S5	P3
S5	P4
S5	P5
S5	P6

$(\pi_{\text{S\#}}(\alpha) \times \beta) - \alpha$

S#	P#
S1	P2
S1	P3
S2	P2
S3	P2
S4	P2
S4	P3

Division (÷) (8 / 9)

And so, finally, we have our answer:

$\pi_{S\#}(\alpha)$

S#
S1
S2
S3
S4
S5

$\pi_{S\#}((\pi_{S\#}(\alpha) \times \beta) - \alpha)$

S#
S1
S2
S3
S4

$\alpha \div \beta = \pi_{S\#}(\alpha) - \pi_{S\#}((\pi_{S\#}(\alpha) \times \beta) - \alpha) =$

S#
S5

Supplier S5 supplies all parts of weight 17 (P2 and P3).

Division (÷) (9 / 9)

Is there a short-cut to avoid that mess? **NO!**

Consider this very similar query:

> Find the S#s of the suppliers that supply
>
> all parts of weight = **19**.

The only weight = 19 part is P6, which this simple expression produces: $\pi_{S\#}(\alpha \bowtie_{P\#} \beta)$ ($\rightarrow$ S4 and S5)

But, when weight = 17, that query gives S2, S3, and S5!