Relational Algebra

Announcements

- Assignment 1 is due, June 16, you have one week extension

Roadmap

- We created 'good' relational schemas applying normalization.
- How do we query the tables to obtain answers?
- We will first define an algebra on sets (well relations are sets!).
- Then we will study an 'implementation' of such an algebra in the form of a query language, called SQL (future lectures).

Why define an Algebra?

- Solid theoretical foundation.
- Algebra consists of primitive operators
  - Think of arithmetic algebra!
- Form queries by combining such operators
  - Focus on implementing each operator independently
  - Allows of optimization of expressions written in the algebra
- Enables reasoning about expressiveness of our expressions

"Core" Relational Algebra

A small set of operators that allow us to manipulate relations in limited but useful ways. The operators are:

1. *Union*, *intersection*, and *difference*
   … the usual set operators!
   – But the relation schemas must be the same.
2. *Selection*: Picking certain rows from a relation
3. *Projection*: Picking certain columns
4. *Products and joins*: Composing relations in useful ways
5. *Renaming* of relations and their attributes

Relational Algebra (cont.)

Characteristics:

- Limited expressive power (subset of possible queries)
- Good optimizer possible
- Rich enough language to express enough useful things

- σ stands for SELECT
- π stands for PROJECT

(UNARY)

- × represents CARTESIAN PRODUCT
- ∪ represents UNION
- – represents SET-DIFFERENCE

(BINARY)

(FUNDAMENTAL)

- ∩ represents SET-INTERSECTION
- $\triangleright\!\triangleleft_\theta$ represents THETA-JOIN
- $\triangleright\!\triangleleft$ represents NATURAL JOIN
- ÷ represents DIVISION or QUOTIENT

Can be defined in terms of the fundamental operations

Union, Intersection, and Difference

R:

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St.	F	9/9/99
Mark Hamill	456 Oak Rd.	M	8/8/88

S:

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St.	F	9/9/99
Harrison Ford	789 Palm Dr.	M	7/7/77

$R \cup S$:

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St.	F	9/9/99
Mark Hamill	456 Oak Rd.	M	8/8/88
Harrison Ford	789 Palm Dr.	M	7/7/77

Union, Intersection, and Difference (cont.)

R:

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St.	F	9/9/99
Mark Hamill	456 Oak Rd.	M	8/8/88

S:

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St.	F	9/9/99
Harrison Ford	789 Palm Dr.	M	7/7/77

$R - S$:

name	*address*	*gender*	*birthdate*
Mark Hamill	456 Oak Rd.	M	8/8/88

$R \cap S$:

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St.	F	9/9/99

Note: $R \cap S = R - (R - S)$

Projection

Syntax: $R_1 = \pi_L(R_2)$

where L is a list of attributes from the schema of R_2

Example:

Movie:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramount	99999

Then the result of executing the query $\pi_{title, length, year}(Movie)$ is:

Result:

title	*year*	*length*
Star Wars	1977	124
Mighty Ducks	1991	104
Wayne's World	1992	95

Another example…

Movie:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramount	99999

Then executing $\pi_{in\text{-}Color}(Movie)$ gives:

Result:

in-Color
true

Selection

Syntax: $R_1 = \sigma_C(R_2)$

where C is a condition involving the attributes of R_2.

Example:

Movie:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramount	99999

Then the result of executing the query $\sigma_{length>=100}(Movie)$ is:

Result:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890

Another example…

Movie:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramount	99999

Then executing $\sigma_{length>=100\ \textbf{AND}\ studio\text{-}Name='Fox'}(Movie)$ gives:

Result:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345

Cartesian Product

Syntax: $R = R_1 \times R_2$

Semantic: pairs each tuple t_1 of R_1 with each tuple t_2 of R_2 and puts in R a tuple t_1t_2

Example:

R

A	B
1	2
3	4

×

S

B	C	D
2	5	6
4	7	8
9	10	11

=

A	R.B	S.B	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

Theta-Join

Syntax: $R = R_1 \bowtie_\theta R_2$

is equivalent to $R = \sigma_\theta(R_1 \times R_2)$

Example:

A	B	C
1	2	3
6	7	8
9	7	8

A<D

B	C	D
2	3	4
2	3	5
7	8	10

=

A	R.B	R.C	S.B	S.C	D
1	2	3	2	3	4
1	2	3	2	3	5
1	2	3	7	8	10
6	7	8	7	8	10
9	7	8	7	8	10

What if we consider $R_{A<D \textbf{ AND } R.B < S.B} S$!?!?

Natural Join

Syntax: $R = R_1 \bowtie R_2$

is equivalent to $R = \pi_{R_1 \cup R_2}(\sigma_C(R_1 \times R_2))$

Semantic: Similar to the theta-join of R_1 and R_2 with the condition that all attributes of the same name be equated. Then, one column for each pair of equated attributes is projected out.

Example:

A	B
1	2
3	4

$\bowtie$

B	C	D
2	5	6
4	7	8
9	10	11

=

A	B	C	D
1	2	5	6
3	4	7	8

Another Example

A	B	C
1	2	3
6	7	8
9	7	8

$\bowtie$

B	C	D
2	3	4
2	3	5
7	8	10

=

A	B	C	D
1	2	3	4
1	2	3	5
6	7	8	10
9	7	8	10

Operator Precedence

The normal way to group operators is:

1. Unary operators σ, π, and ρ have highest precedence.
2. Next highest are the "multiplicative" operators like join, theta join and cross product.
3. Lowest are the "additive" operators union, intersection and difference.

But there is no universal agreement, so we always put parentheses *around* the argument of a unary operator, and it is a good idea to group all binary operators with parentheses *enclosing* their arguments.

Example:

Group $R \cup \sigma\ S$ *join* T as $R \cup (\sigma(S)$ join $T)$.

Each Expression Needs a Schema

- If union, intersection and difference are applied then the resulting schema is the same with the schema(s) of the arguments.
- Projection: use the attributes listed in the projection.
- Selection: no change in schema.
- Product $R \times S$: use attributes of R and S.
  - But if they share an attribute A, prefix it with the relation name, as $R.A$, $S.A$.
- Theta-join: same as product.
- Natural join: use attributes from each relation; common attributes are merged anyway.
- Renaming: whatever it says.

Expression trees

Consider the decomposed relations:

Movies$_1$(*title*, *year*, *length*, *film-type*, *studio-Name*)

Movies$_2$(*title*, *year*, *star-Name*)

Query: "What are the titles and years of movies made by Fox that are at least 100 minutes long"

Answer: $\pi_{title,\ year}(\sigma_{studio\text{-}Name=\text{"Fox"}}(Movies_1) \cup \sigma_{length >= 100}(Movies_1))$

Expression trees (cont.)

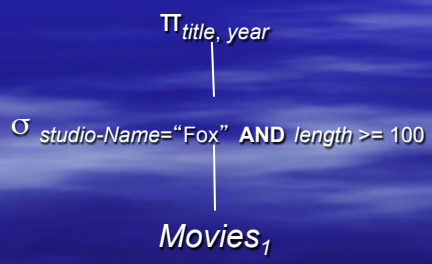
Consider the decomposed relations:

Movies$_1$(*title*, *year*, *length*, *film-type*, *studio-Name*)

Movies$_2$(*title*, *year*, *star-Name*)

Query: "What are the titles and years of movies made by Fox that are at least 100 minutes long"

Answer: $\pi_{title,\ year}(\sigma_{studio\text{-}Name=\text{"Fox"}}$ **AND** $length >= 100(Movies_1))$



Expression trees (cont.)

Consider the decomposed relations:

Movies$_1$(*title*, *year*, *length*, *film-type*, *studio-Name*)

Movies$_2$(*title*, *year*, *star-Name*)

Query: "Find the stars of movies that are at least 100 mins long"

Answer: $\pi_{star\text{-}Name}(\sigma_{length >= 100}(Movies_1 \bowtie Movies_2))$

Expression trees (cont.)

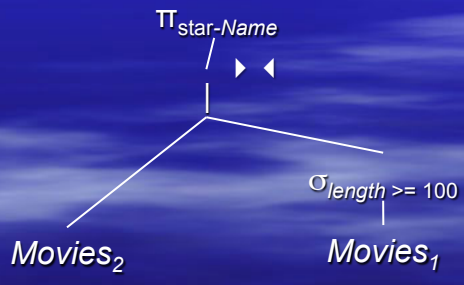
Consider the decomposed relations:

Movies$_1$(*title*, *year*, *length*, *film-type*, *studio-Name*)

Movies$_2$(*title*, *year*, *star-Name*)

Query: "Find the stars of movies that are at least 100 mins long"

Answer: $\pi_{star\text{-}Name}(Movies_2 \bowtie (\sigma_{length >= 100}(Movies_1)))$

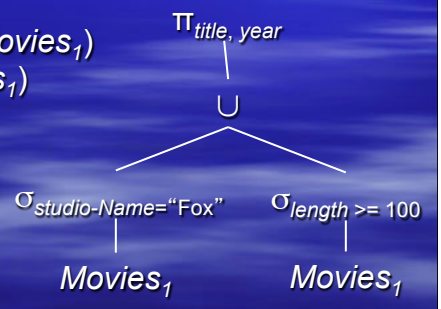


Linear Notation for Expressions

- Invent new names for intermediate relations, and assign them values that are algebraic expressions.
- Renaming of attributes implicit in schema of new relation.

Example:

$Tmp_1(t, y, l, f, s) = \sigma_{studio\text{-}Name=\text{“Fox”}}(Movies_1)$
$Tmp_2(t, y, l, f, s) = \sigma_{length >= 100}(Movies_1)$
$Tmp_3(t, y, l, f, s) = Tmp_1 \cup Tmp_2$
$Answer(title, year) = \pi_{t,y}(Tmp_3)$



Bag Semantics

A relation (in SQL, at least) is really a *bag* or a *multiset*.

- It may contain the same tuple more than once, although there is no specified order (unlike a list).
- Example: {1,2,1,3} is a bag and not a set.
- Select, project, and join work for bags as well as sets.
  - Just work on a tuple-by-tuple basis, and don't eliminate duplicates.

Bag Union/Intersection/Difference

Union: Sum the times an element appears in the two bags.

- Example: {1,2,1} ∪ {1,2,3,3} = {1,1,1,2,2,3,3}

Intersection: Take the minimum of the number of occurrences in each bag.

- Example: {1,2,1} ∩ {1,2,3,3} = {1,2}.

Difference: Proper-subtract the number of occurrences in the two bags.

- Example: {1,2,1} – {1,2,3,3} = {1}.

Laws for Bags / Laws for Sets

- Some familiar laws continue to hold for bags.
  - Examples: union and intersection are still commutative and associative.
- But other laws that hold for sets do not hold for bags.

Example:

$R \cap (S \cup T) \equiv (R \cap S) \cup (R \cap T)$ holds for sets.

- Let R, S, and T each be the bag {1}.
- Left side: $S \cup T$ = {1,1}; $R \cap (S \cup T)$ = {1}.
- Right side: $R \cap S = R \cap T$ = {1};
  $(R \cap S) \cup (R \cap T)$ = {1} ∪ {1} = {1,1} ≠ {1}.

Extended ("Non-classical") Relational Algebra

Adds features needed for SQL, bags.

1. Duplicate-elimination operator δ.
2. Extended projection.
3. Sorting operator τ.
4. Grouping-and-aggregation operator γ.
5. Outerjoin operator join_o

Duplicate Elimination

Syntax: $R_2 = \delta(R_1)$

Semantics: Relation R_2 has one copy of each tuple that appears one or more times in relation R_1.

Example:

R_1:

A	B
1	2
3	4
1	2
1	2

$\delta(R_1)$:

A	B
1	2
3	4

Sorting

Syntax: $\tau_L(R)$

Semantic: It returns a list of tuples of R, ordered according to attributes on list L.

Note: The result type is outside the normal types (set or bag) for relational algebra.

Example:

R:

A	B
1	3
3	4
5	2

$\tau_B(R)$ = [(5,2), (1,3), (3,4)]

Extended Projection

Syntax: $R_1 = \pi_L(R_2)$

where L is a list of

- attributes from the schema of R_2
- expressions of the form $x \rightarrow y$, where x, y are attributes of R_2
- expressions of the form $E \rightarrow z$, where E is a *formula* and z a name

Example:

R:

A	B
1	2
3	4

$\pi_{A+B \rightarrow C, A \rightarrow D, A \rightarrow E}(R)$:

C	D	E
3	1	1
7	3	3

Aggregation Operators

- These are not relational operators; rather they summarize a column in some way.
- Five standard operators: SUM, AVG, MIN, MAX and COUNT.

Example:

R:

A	*B*
1	2
3	4
1	2
1	2

- SUM(*B*) = 2 + 4 + 2 + 2 = 10
- AVG(*A*) = (1 + 3 + 1 + 1) / 4 = 1.5
- MIN(*A*) = 1
- MAX(*A*) = 3
- COUNT(*A*) = 4

Grouping Operator

Syntax: $\gamma_L(R)$

where *L* is a list of elements that are either

- Individual (*grouping*) attributes, or
- Of the form $\theta(A)$, where θ is an aggregation operator and *A* the attribute to which it is applied

Semantics: $\gamma_L(R)$ is computed by:

1. Group *R* according to all the grouping attributes on list *L*.
2. Within each group, compute $\theta(A)$, on all elements of the group.
3. Result is the relation whose columns consist of one tuple for each group. The components of that tuple are the values associated with each element of *L* for that group and the aggregations over all tuples of that group for the aggregated attributes on list L.

Example

Movie:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramount	99999
Spider-Man	2002	121	true	Columbia	12345
Episode I	1999	133	true	Fox	45634
Episode II	2002	142	true	Fox	23456

Compute $\gamma_{studio\text{-}Name,AVG(length)}(Movie)$

1. Group by the grouping attribute(s), *studio-Name* in this case:

Star Wars	1977	124	true	Fox	12345
Episode I	1999	133	true	Fox	45634
Episode II	2002	142	true	Fox	23456
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramount	99999
Spider-Man	2002	121	true	Columbia	12345

Example (cont.)

2. Compute average of length within groups:

studio-Name	AVG(*length*)
Fox	133
Disney	104
Paramount	95
Columbia	121

Outer-join

The normal join can "lose" information, because a tuple that doesn't join with any from the other relation

(*dangles*) has no vestige in the join result.

- The *null value* ⊥ can be used to "pad" dangling tuples so that they appear in the join
- Gives us the *outer-join* operator join_o
- Variations: theta-outer-join, left- and right-outer-join (pad only dangling tuples from the left (respectively, right).

Example

R:

A	*B*
1	2
3	4

S:

B	*C*
4	5
6	7

$R \blacktriangleright\!\blacktriangleleft_o S$:

A	*B*	*C*
3	4	5
1	2	⊥
⊥	6	7

$R \blacktriangleright\!\blacktriangleleft_{oL} S$:

A	*B*	*C*
3	4	5
1	2	⊥

$R \blacktriangleright\!\blacktriangleleft_{oR} S$:

A	*B*	*C*
3	4	5
⊥	6	7

Division operator

- Not a primitive operator but useful for queries like
  - Find students who have passed ALL courses
- Let A have 2 fields x and y; B have field y
  - A/B = {<x> | exists <x,y> in A forall <y> in B}
  - A/B contains all x tuples such that for every y tuple in B there is an xy tuple in A
  - Or if the set of y values associated with an x value in A contains all y values in B the x value is in A/B

examples

A

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

B1

pno
p2

A/B1

sno
s1
s2
s3
s4

B2

pno
p2
p4

A/B2

sno
s1
s4

B3

pno
p1
p2
p4

A/B3

sno
s1

Expressing A/B

- Division is not essential op; just a useful shorthand.
- For A/B compute all x values that are not disqualified by some y value in B
  - X value is disqualified if by attaching y value from B we obtain an xy tuple that is not in A
    - $\Pi_x((\pi_x(A) \times B) - A)$
  - A/B $\pi_x(A)$- all disqualified tuples

That's it for today…