VirginiaTech

CS 4604: Introduction to Database Management Systems

B. Aditya Prakash

Lecture #3: SQL and Relational Algebra---Part 1

Reminder: Relational Algebra

- Relational algebra is a notation for specifying queries about the contents of relations
- Notation of relational algebra eases the task of reasoning about queries
- Operations in relational algebra have counterparts in SQL

What is an Algebra?

- An algebra is a set of operators and operands
  - Arithmetic: operands are variables and constants, operators are +,−,×,÷, /, etc.
  - Set algebra: operands are sets and operators are ∩, U, -

- An algebra allows us to
  - **construct expressions** by combining operands and expression using operators
  - has **rules for reasoning** about expressions

$a^2 + 2 \times a \times b + 2b, \quad (a + b)^2$

$R - (R - S), \quad R \cap S$

Relational operators

- selection $\sigma_{condition}(R)$
- Projection $\pi_{att-list}(R)$

- set union R U S
- set difference R - S

Clarification: Projection

- The projection operator produces from a relation R a new relation containing only **some of R's columns**

- "Delete" (i.e. not show) attributes not in projection list
- Duplicates eliminated (sets vs *multisets*)

- To obtain a relation containing only the columns $A_1, A_2, \ldots A_n$ of R

RA: $\pi_{A_1, A_2, \ldots A_n}(R)$

SQL: **SELECT** $A_1, A_2, \ldots A_n$ **FROM** R;

VirginiaTech

Clarification: Projection

- The projection operator produces from a relation R a new relation containing only **some of R's columns**

- "Delete" (i.e. not show) attributes not in projection list
- Duplicates eliminated (sets vs *multi*sets)

- To obtain a relation containing only the columns $A_1, A_2, \ldots A_n$ of R

RA: $\pi_{A_1, A_2, \ldots A_n}(R)$

SQL: **SELECT DISTINCT** $A_1, A_2, \ldots A_n$ **FROM** R;

What about strings?

find student ssns who live on “main” (st or str or street - ie., “main st” or “main str” …)

What about strings?

find student ssns who live on "main" (st or str or street)

select ssn

from student

where address **like** "main%"

%: variable-length don't care

_: single-character don't care

Relational operators

Are we done yet?

Q: Give a query we can **not** answer yet!

Relational operators

A: any query across **two** or more tables,

eg., 'find names of students in 4604'

Q: what extra operator do we need??

STUDENT		
Ssn	Name	Address
123	smith	main str
234	jones	forbes ave

SSN	c-id	grade
123	4604	A
234	3114	B

Relational operators

A: any query across **two** or more tables,

eg., 'find names of students in 4604'

Q: what extra operator do we need??

A: surprisingly, cartesian product is enough!

STUDENT		
Ssn	Name	Address
123	smith	main str
234	jones	forbes ave

SSN	c-id	grade
123	4604	A
234	3114	B

Cartesian Product

- The Cartesian product (or cross-product or product) of two relations R and S is a the set of pairs that can be formed by pairing each tuple of R with each tuple of S.
  - The result is a relation whose schema is the schema for R followed by the schema for S.

RA: R **X** S

SQL: **SELECT** * **FROM** R , S ;

Cartesian Product

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1 X R1

?

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

We rename attributes to avoid ambiguity or we prefix attribute with the name of the relation it belongs to.

FUNDAMENTAL Relational operators

- selection $\sigma_{condition}(R)$
- projection $\pi_{att-list}(R)$
- cartesian product R X S
- set union R U S
- set difference R - S

Relational ops

- Surprisingly, they are enough, to help us answer almost any query we want!
- derived/convenience operators:
  - set intersection ---(We have seen this)
  - **join** (theta join, equi-join, natural join) $\bowtie$
  - 'rename' operator $\rho_{R'}(R)$
  - division $R \div S$

VirginiaTech

Theta-Join

- The theta-join of two relations R and S is the set of tuples in the Cartesian product of R and S that satisfy some condition C.

RA: $R \bowtie_C S$

SQL:

SELECT *
FROM R, S
WHERE C;

- $R \bowtie_C S = \sigma_C(R \times S)$

Theta-Join

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

$$R \bowtie_{c} S = \sigma_{c}(R \times S)$$

Natural Join

- The natural join of two relations R and S is a set of pairs of tuples, one from R and one from S, that agree on whatever attributes are common to the schemas of R and S.
- The schema for the result contains the **union of the attributes of R and S.** (so duplicate cols. are dropped)
- Assume the schemas R(A,B, C) and S(B, C,D)

RA: $R \bowtie S$

SQL: **SELECT** R.A, R.B, R.C, S.D
FROM R, S
WHERE R.B = S.B **AND** R.C = S.C;

Natural Join: Nit-picking

- What if R and S have not attributes in common?

natural join → cartesian product

- Some (like Oracle) provide a special single NATURAL JOIN operator, but some (like IBM DB2) don't.
  - So assume there is no special **SQL** natural join operator

Operators so far

- **Remove parts of single relations**
  - Projection: $\pi_{(A,B)}(R)$ and SELECT A, B FROM R
  - Selection: $\sigma_C(R)$ and SELECT * FROM R WHERE C
  - Combining Projection and Selection:
  - $\pi_{(A,B)}(\sigma_C(R))$
  - SELECT A, B FROM R WHERE C

VirginiaTech

Operations so far

- **Set operations**
  - R and S must have the same attributes, same attribute types, and same order of attributes
  - Union: R U S and (R) UNION (S)
  - Intersection: R ∩ S and (R) INTERSECT (S)
  - Difference: R – S and (R) EXCEPT (S)

Operations so far

- **Combine the tuples of two relations**
  - Cartesian Product: R X S, …. FROM R, S …..
  - Theta Join: R $\bowtie_C$ S, …. FROM R, S WHERE C
  - Natural Join: R $\bowtie$ S

Ordering

- find student records, sorted in name order
  - select *
  - from student
  - order by name asc

  - asc is the default

Ordering

- find student records, sorted in name order; break ties by reverse ssn
  – select *
  – from student
  – order by name, ssn desc

Rename op.

- Q: why? $\rho_{AFTER}(BEFORE)$
- A: shorthand; self-joins; …
- for example, find the grand-parents of ‘Tom’, given PC (parent-id, child-id)

Rename op.

- PC (parent-id, child-id)

$$PC \bowtie PC$$

PC	
p-id	c-id
Mary	Tom
Peter	Mary
John	Tom

PC	
p-id	c-id
Mary	Tom
Peter	Mary
John	Tom

Rename op.

- first, WRONG attempt:

$$PC \bowtie PC$$

(why? how many columns?)

- Second WRONG attempt:

$$PC \bowtie_{PC.c-id=PC.p-id} PC$$

Rename op.

- we clearly need two different names for the same table - hence, the 'rename' op.

$$\rho_{PC1}(PC) \bowtie_{PC1.c-id=PC.p-id} PC$$

Disambiguation and Renaming

RA: give R the name S;

R has n attributes,

which are $\rho_{S\ (A1, A2, \ldots\ An)}\ (R)$

called A1, A2, . . . , An in S

SQL: Use the **AS** keyword in the **FROM** clause:
Students AS Students1 renames Students to Students1.

SQL: Use the **AS** keyword in the **SELECT** clause to rename attributes.

Disambiguation and Renaming

- Name pairs of students who live at the same address: Students (Name, Address)

RA: $$\pi_{S1.Name,S2.Name}(\sigma_{S1.Address=S2.Address}(\rho_{S1}(Students) \times \rho_{S2}(Students)))$$

SQL:
```
SELECT S1.name, S2.name
FROM Students AS S1, Students AS S2
WHERE S1.address = S2.address
```

Disambiguation and Renaming

- Name pairs of students who live at the same address:

SQL: SELECT S1.name, S2.name
FROM Students AS S1, Students AS S2
WHERE S1.address = S2.address

- Are these correct?
- No !!! the result includes tuples where a student is paired with himself/herself
- Solution: Add the condition S1.name <> S2.name.

Division

- Rarely used, but powerful.
- Example: find suspicious suppliers, ie., suppliers that supplied all the parts in A_BOMB

Division

SHIPMENT	
s#	p#
s1	p1
s2	p1
s1	p2
s3	p1
s5	p3

÷

ABOMB
p#
p1
p2

=

BAD_S
s#
s1

Division

- Observations: ~reverse of cartesian product
- It can be derived from the 5 fundamental operators (!!)
- How?

Division

- Answer:

$$r \div s = \pi_{(R-S)}(r) - \pi_{(R-S)}[(\pi_{(R-S)}(r) \times s) - r]$$

- Observation: find 'good' suppliers, and subtract! (**double negation**)

Division

- Answer:

r

SHIPMENT	
s#	p#
s1	p1
s2	p1
s1	p2
s3	p1
s5	p3

÷

s

ABOMB
p#
p1
p2

=

BAD_S
s#
s1

$$r \div s = \pi_{(R-S)}(r) - \pi_{(R-S)}[(\pi_{(R-S)}(r) \times s) - r]$$

R: attributes of r
S: attributes of s

- Observation: find 'good' suppliers, and subtract! (**double negation**)

Division

- Answer:

r

SHIPMENT	
s#	p#
s1	p1
s2	p1
s1	p2
s3	p1
s5	p3

÷

s

ABOMB
p#
p1
p2

=

BAD_S
s#
s1

$$r \div s = \pi_{(R-S)}(r) - \pi_{(R-S)}[(\pi_{(R-S)}(r) \times s) - r]$$

All suppliers

All bad parts

Division

- Answer:

SHIPMENT	
s#	p#
s1	p1
s2	p1
s1	p2
s3	p1
s5	p3

÷

ABOMB
p#
p1
p2

=

BAD_S
s#
s1

$$r \div s = \pi_{(R-S)}(r) - \pi_{(R-S)}[(\pi_{(R-S)}(r) \times s) - r]$$

all possible
suspicious shipments

Division

- Answer:

SHIPMENT	
s#	p#
s1	p1
s2	p1
s1	p2
s3	p1
s5	p3

÷

ABOMB
p#
p1
p2

=

BAD_S
s#
s1

$$r \div s = \pi_{(R-S)}(r) - \pi_{(R-S)}[(\pi_{(R-S)}(r) \times s) - r]$$

all possible
suspicious shipments
that didn't happen

Division

- Answer:

SHIPMENT	
s#	p#
s1	p1
s2	p1
s1	p2
s3	p1
s5	p3

$\div$

ABOMB
p#
p1
p2

$=$

BAD_S
s#
s1

$$r \div s = \pi_{(R-S)}(r) - \pi_{(R-S)}[(\pi_{(R-S)}(r) \times s) - r]$$

all suppliers who missed
at least one suspicious shipment,
i.e.: 'good' suppliers

VirginiaTech

Quick Quiz: Independence of Operators

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

$$R \bowtie_C = \sigma_C(R \times S)$$

$$R \bowtie S = ??$$

Quick Quiz: Independence of Operators

$$R \bowtie S$$

- Suppose R and S share the attributes A1,A2,..An
- Let L be the list of attributes in R \Union list of attributes in S (so no duplicate attributes)
- Let C be the condition

R.A1 = S.A1 *AND* R.A2 = S.A2 *AND* ..... R.An = S.An

$$R \bowtie S = \pi_L(\sigma_C(R \times S))$$

Linear Notation for Relational Algebra

- Relational algebra expressions can become very long.

- Use linear notation to store results of intermediate expressions.
  - A relation name and a parenthesized list of attributes for that relation
  - Use Answer as the conventional name for the final result
  - The assignment symbol :=
  - Any expression in relational algebra on the right

Example of Linear Notation

- Name pairs of students who live at the same address.
- Normal expression:

$$\pi_{S1.Name, S2.Name}(\sigma_{S1.Address=S2.Address}(\rho_{S1}(Students) \times \rho_{S2}(Students)))$$

Example of Linear Notation

- Normal expression:

$$\pi_{S1.Name,S2.Name}($$
$$\sigma_{S1.Address=S2.Address}$$
$$(\rho_{S1}(Students) \times \rho_{S2}(Students)))$$

- Linear Notation:

$$\text{Pairs}(\text{P1, N1, A1, P2, N2, A2}) := \rho_{S1}(Students) \times \rho_{S2}(Students)$$
$$\text{Matched}(\text{P1, N1, A1, P2, N2, A2}) :=$$
$$\sigma_{A1=A2}(\text{Pairs}(\text{P1, N1, A1, P2, N2, A2}))$$
$$\text{Answer}(\text{Name1, Name2}) := \pi_{N1,N2}(\text{Matched}(\text{P1, N1, A1, P2, N2, A2}))$$

Interpreting Queries Involving Multiple Relations

- SELECT A, B FROM R, S WHERE C;
- Nested loops:

```
for each tuple t1 in R
    for each tuple t2 in S
        if the attributes in t1 and t2 satisfy C
            output the tuples involving attributes A and B
```

Interpreting Queries Involving Multiple Relations

- SELECT A, B FROM R, S WHERE C;
- Conversion to relational algebra:

$$\pi_{A,B}(\sigma_C(R \times S))$$

Compute R X S

Apply selection operator σ() to R X S

Project the result tuples to attributes A and B

Aggregate functions

- find avg grade, across all students

  select ??

  from takes

takes

SSN	c-id	grade
123	15-413	4
234	15-413	3

Aggregate functions

- find avg grade, across all students

  select avg(grade)

  from takes

takes

SSN	c-id	grade
123	15-413	4
234	15-413	3

- result: a single number
- Which other functions?

Aggregate Operators

- *COUNT (*)*
- *COUNT ([DISTINCT] A)*
  - A is a column
- *SUM ([DISTINCT] A)*
- *AVG ([DISTINCT] A)*
- *MAX (A)*
- *MIN (A)*

Aggregate functions

- find total number of enrollments

```
select count(*)
from takes
```

takes

SSN	c-id	grade
123	15-413	4
234	15-413	3

VirginiaTech

Aggregate functions

- find total number of students in 15-413

  select count(*)

  from takes

  where c-id="15-413"

takes

SSN	c-id	grade
123	15-413	4
234	15-413	3

Find name and age of the oldest sailor(s)

Sailors

sid	name	age	ratings
45	Tom	34	5.0
…	…..	……	……

SELECT S.sname, MAX (S.age)
FROM Sailors S

- This is illegal, but why?
  - Cannot combine a column with a value

SELECT S.sname, S.age
FROM Sailors S
WHERE S.age = (SELECT MAX (S2.age) FROM Sailors S2)

GROUP BY and HAVING

- So far, aggregate operators are applied to all (qualifying) tuples.
  - Can we apply them to each of several groups of tuples?
- Example: find the age of the youngest sailor for each rating level.
  - In general, we don't know how many rating levels exist, and what the rating values for these levels are!
  - Suppose we know that rating values go from 1 to 10; we can write 10 queries that look like this:

For i = 1, 2, ... , 10:

```
SELECT  MIN (S.age)
FROM  Sailors S
WHERE  S.rating = i
```

Find the age of the youngest sailor for each rating level

Sid	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.5
85	Art	3	25.5
32	Andy	8	25.5
95	Bob	3	63.5

```
SELECT S.rating, MIN (S.age) as age
FROM Sailors S
GROUP BY  S.rating
```

(1) The sailors tuples are put into "same rating" groups.

(2) Compute the Minimum age for each rating group.

(1)

Rating	Age
3	25.5
3	63.5
7	45.0
8	55.5
8	25.5

(2)

Rating	Age
3	25.5
7	45.0
8	25.5

Find the age of the youngest sailor for each rating level that **has at least 2 members**

Sid	Sname	Rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.5
85	Art	3	25.5
32	Andy	8	25.5
95	Bob	3	63.5

SELECT S.rating, MIN (S.age) as minage
FROM Sailors S
GROUP BY S.rating
HAVING COUNT(*) > 1

1. The sailors tuples are put into "same rating" groups.
2. Eliminate groups that have < 2 members.
3. Compute the Minimum age for each rating group.

Rating	Age
3	25.5
3	63.5
7	45.0
8	55.5
8	25.5

Rating	Minage
3	25.5
8	25.5

Drill

- find total number of students in each course

SSN	c-id	grade
123	15-413	4
234	15-413	3

c-id	count
15-413	2

Drill

- find total number of students in each course

  select c-id, count(*)

  from takes

  group by c-id

  order by c-id

SSN	c-id	grade
123	15-413	4
234	15-413	3

c-id	count
15-413	2

Drill

- find total number of students in each course, and sort by count, decreasing

SSN	c-id	grade
123	15-413	4
234	15-413	3

c-id	pop
15-413	2

Drill

- find total number of students in each course, and sort by count, decreasing

  select c-id, count(*) as pop

  from takes

  group by c-id

  order by pop desc

SSN	c-id	grade
123	15-413	4
234	15-413	3

c-id	pop
15-413	2

Queries With *GROUP BY* and *HAVING*

SELECT [DISTINCT] target-list
FROM relation-list
WHERE qualification
GROUP BY grouping-list
HAVING group-qualification

SELECT S.rating, MIN (S.age) as age
FROM Sailors S
GROUP BY S.rating
HAVING S.rating > 5

- The *target-list* contains (i) attribute names (ii) terms with aggregate operations (e.g., AVG (*S.age*)).
- The attribute list (e.g., *S.rating*) in *target-list* must be in *grouping-list*.
- The attributes in group-qualification must be in *grouping-list*.

Motivation for Subqueries

- Find the name of the professor who teaches "CS 4604."

```
SELECT Name
FROM Professors, Teach
WHERE (PID = ProfessorPID) AND (Number =
'4604') AND (DeptName = 'CS');
```

- Do we need to take the natural join of two big relations just to get a relation with one tuple?
- Can we rewrite the query without using a join?

Nesting

- A query can be put inside another query
- Most commonly in the WHERE clause
- Sometimes in the FROM clause (depending on the software)
- This subquery is executed first (if possible)