Relational Algebra and Calculus

Yanlei Diao & Michaël Thomazo

Relational Query Languages

- Relational model allows simple, powerful *querying* of data.
- Relational query languages:
  - Declarative: say "what you want", not "how to get it"
  - Formal mathematical model
  - Query optimization

Formal Relational Query Languages

- Two mathematical languages form the basis for the "real" one, SQL, and for implementation:
  - *Relational Algebra*: operational, useful for representing execution plans.
  - *Relational Calculus*: declarative, useful for defining query semantics.

Outline

- Relational Model
- Formal Query Languages
  - Relational Algebra
  - Relational Calculus
  - Language Theory

What is an "Algebra"?

- A mathematical system consisting of:
  - Operands: variables or values from which new values can be constructed
  - Operators: symbols denoting procedures that construct new values from given values

What is "Relational Algebra"

- Relational algebra:
  - Operands are relations.
  - Operators each take 1 or 2 relations and produce a relation.
- Closure property: relational algebra is <u>closed</u> under the relational model.
  - Relational operators can be arbitrarily *composed*!

Relational Algebra

- Basic operations:
  - *Selection* (σ) Selects a subset of rows from a relation.
  - *Projection* (π) Deletes unwanted columns from a relation.
  - *Cross-product* ($\times$) Allows us to combine two relations.
  - *Set-difference* ($-$) Tuples in reln. 1, but not in reln. 2.
  - *Union* ($\cup$) Tuples in reln. 1 and in reln. 2.
- Additional operations:
  - *Join* ($\bowtie$), *Intersection* ($\cap$), *Division* ($/$), *Renaming* (ρ)
  - Can be derived from basic operators. Not essential, but useful!

Example Instances

Sailors

S1

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

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Reserves

R1

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

Projection

- Retain only attributes in the *projection list*; delete others.
- *Schema of result* contains exactly the fields in projection list.

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$$\pi_{sname,rating}(S2)$$

Projection (contd.)

- Projection operator has to eliminate *duplicates*!
  - Real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

age
35.0
55.5

$$\pi_{age}(S2)$$

Selection

- Select rows that satisfy the *selection condition*; discard others.
- *Schema of result* identical to schema of input.

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$$\sigma_{rating>8}(S2)$$

Selection (contd.)

- *Composition*: *result* relation of an operator can be the *input* to another operator.

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$$\sigma_{rating>8}(S2)$$

sname	rating
yuppy	9
rusty	10

$$\pi_{sname,rating}(\sigma_{rating>8}(S2))$$

Union, Intersection, Set-Difference

- Set operations:
  - Union (∪)
  - Intersection (∩)
  - Set difference (-)
- Take two input relations, which must be *union-compatible*:
  - Same number of fields.
  - Corresponding fields have the the same type.
- What is the *schema* of result?

S1

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

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Example Set Operations

S1

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

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

Duplicate elimination: remove tuples that have same values in all attributes.

Example Set Operations

S1

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

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	sname	rating	age
22	dustin	7	45.0

$$S1 - S2$$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$$S1 \cap S2$$

Duplicates?

Cross (Cartesian) Product

- S1 × R1: Each row of S1 is paired with each row of R1.

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* × *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

Cross-Product (contd.)

(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

- *Result schema* inherits all fields of S1 and R1.
  - *Conflict*: Both S1 and R1 have a field called *sid*.
- *Renaming operator*: $\rho\ (C(1 \rightarrow sid1, 5 \rightarrow sid2),\ S1 \times R1)$

Joins

- Condition (theta) Join: $R \bowtie_c S = \sigma_c (R \times S)$

(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

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

  - *Result schema* same as that of cross-product.
  - But often fewer tuples, more efficient for computation.

Joins

- *Equi-Join*: A special case of condition join where the condition θ contains only *equalities*.

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

$$S1 \bowtie_{sid} R1$$

  - *Result schema* contains only *one copy* of fields for which equality is specified.

- *Natural Join* ($R \bowtie S$): equijoin on *all* common fields.

Example Schema

- **Sailors**(*sid*: integer, *sname*: string, *rating*: integer, *age*: integer)
- **Boats**(*bid*: integer, *color*: string)
- **Reserves**(*sid*: integer, *bid*: integer, *day*: date)

Find names of sailors who've reserved boat #103

- **Sailors**(*sid*: integer, *sname*: string, *rating*: integer, *age*: integer)
- **Boats**(*bid*: integer, *color*: string)
- **Reserves**(*sid*: integer, *bid*: integer, *day*: date)

Find names of sailors who've reserved boat #103

- Solution 1: $\pi_{sname}((\sigma_{bid=103}\text{Reserves})\bowtie Sailors)$

- Solution 2: $\rho\ (Temp1,\ \sigma_{bid=103}\text{Reserves})$

  $\rho\ (Temp2,\ Temp1 \bowtie Sailors)$

  $\pi_{sname}(Temp2)$

- Solution 3: $\pi_{sname}(\sigma_{bid=103}(\text{Reserves}\bowtie Sailors))$

Algebraic equivalence!

Find names of sailors who've reserved a red boat

- **Sailors**(*sid*: integer, *sname*: string, *rating*: integer, *age*: integer)
- **Boats**(*bid*: integer, *color*: string)
- **Reserves**(*sid*: integer, *bid*: integer, *day*: date)

- Boat color is only available in Boats; so need an extra join:

$$\pi_{sname}((\sigma_{color='red'} Boats) \bowtie \text{Re}serves \bowtie Sailors)$$

Find sailors who've reserved a red and a green boat

- Will a single selection work?
- Instead, *intersect* sailors who' ve reserved red boats and sailors who' ve reserved green boats.

sid is a *key* for Sailors

$$\rho\ (Tempred, \pi_{sid}((\sigma_{color='red'} Boats) \bowtie Reserves))$$

$$\rho\ (Tempgreen, \pi_{sid}((\sigma_{color='green'} Boats) \bowtie Reserves))$$

$$\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$$

Outline

- Relational Model
- Formal Query Languages
  - Relational Algebra
  - Relational Calculus
  - Language Theory

Relational Calculus

- *Query* has the form:

$$\left\{ \langle x1, x2, \ldots, xn \rangle \mid p\left(\langle x1, x2, \ldots, xn \rangle \right) \right\}$$

domain variables, or *constants*

formula

- *Answer* includes all *tuples* $\langle x1, x2, \ldots, xn \rangle$ that make the formula $p\left(\langle x1, x2, \ldots, xn \rangle \right)$ *true*.

Formulas

- *Formula* is recursively defined:
  - *Atomic formulas*: getting tuples from relations, or making comparisons of values
  - *Logical connectives*: $\neg$, $\wedge$, $\vee$
  - *Quantifiers*: $\exists$, $\forall$

Free and Bound Variables

- The use of quantifiers $\exists X$ and $\forall X$ in a formula is said to *bind* X.
  - A variable that is not bound is free.
- Let us revisit the definition of a query:

$$\{\langle x1, x2, \ldots, xn \rangle \mid p(\langle x1, x2, \ldots, xn \rangle)\}$$

- There is an important restriction: the variables x1, ..., xn that appear to the left of \`|' must be the *only* free variables in the formula p(...).

Find sailors rated > 7 who have reserved boat #103

Relational Algebra:

$$\pi_{sname}((\sigma_{bid=103}\text{Reserves}) \bowtie (\sigma_{rating>7}Sailors))$$

Relational Calculus:

$$\{X_{\text{sname}} \mid \exists\, X_{\text{sid}}, X_{\text{rating}}, X_{\text{age}}\ Sailors(X_{\text{sid}}, X_{\text{sname}}, X_{\text{rating}}, X_{\text{age}}) \wedge X_{\text{rating}} > 7$$
$$\wedge\ \exists\, X_{\text{bid}}, X_{\text{day}}\ Reserves(X_{\text{sid}}, X_{\text{bid}}, X_{\text{day}}) \wedge X_{\text{bid}}{=}103\ \}$$

- Where is the join?
  - Use ∃ to find a tuple in Reserves that `joins with' the Sailors tuple under consideration.

Find names of sailors who've reserved all boats

$$
\begin{aligned}
\{X_{sname} \mid \exists\, X_{sid}, X_{rating}, X_{age}\ &\langle X_{sid}, X_{sname}, X_{rating}, X_{age}\rangle \in Sailors \land \\
&\forall\, \langle X_{bid}, X_{color}\rangle \in Boats \\
&\quad (\exists\, X_{day}\ \langle X_{sid}, X_{bid}, X_{day}\rangle \in Reserves) \quad \}
\end{aligned}
$$

- To find sailors who've reserved *all red* boats:

$$
\begin{aligned}
\{X_{sname} \mid \exists\, X_{sid}, X_{rating}, X_{age}\ &\langle X_{sid}, X_{sname}, X_{rating}, X_{age}\rangle \in Sailors \land \\
&\forall\, \langle X_{bid}, X_{color}\rangle \in Boats \\
&\quad (X_{color} \neq \text{'red'} \lor \exists\, X_{day}\ \langle X_{sid}, X_{bid}, X_{day}\rangle \in Reserves)\ \}
\end{aligned}
$$

Find names of sailors who've reserved all boats

$$\{X_{sname} \mid \exists\, X_{sid}, X_{rating}, X_{age}\, \langle X_{sid}, X_{sname}, X_{rating}, X_{age}\rangle \in Sailors \wedge$$
$$\forall\, \langle X_{bid}, X_{color}\rangle \in Boats$$
$$(\exists\, X_{day}\, \langle X_{sid}, X_{bid}, X_{day}\rangle \in Reserves) \quad \}$$

$$\Downarrow$$

$$\{X_{sname} \mid \exists\, X_{sid}, X_{rating}, X_{age}\, \langle X_{sid}, X_{sname}, X_{rating}, X_{age}\rangle \in Sailors \wedge$$
$$\neg\exists\, \langle X_{bid}, X_{color}\rangle \in Boats$$
$$(\neg\exists\, X_{day}\, \langle X_{sid}, X_{bid}, X_{day}\rangle \in Reserves) \quad \}$$

Find the names of sailors who've reserved all boats

- Step 1: for each sailor, check if there exists a boat that he has not reserved (called formula F).

$$\rho(S\_neg, \pi_{sid}((\pi_{sid} Reserves) \times (\pi_{bid} Boats) - (\pi_{sid,bid} Reserves)))$$

- Step 2: find sailors for which F is not true and retrieve their names

$$\pi_{sname}((\pi_{sid} Reserves - S\_neg) \bowtie Sailors)$$

'–' : the only way to express negation in relational algebra!

Outline

- Relational Model
- Formal Query Languages
  - Relational Algebra
  - Relational Calculus
  - Language Theory

Query Languages

- The three languages we consider:
  - Relational Algebra (RA)
  - Relational Calculus (RC)
  - Structured Query Language (SQL)

Find sailors rated > 7 who have reserved boat #103

Relational Algebra:

$$\pi_{sname}((\sigma_{bid=103} Reserves) \bowtie (\sigma_{rating>7} Sailors))$$

Relational Calculus:

$$\{X_{\text{sname}} \mid \exists\, X_{\text{sid}}, X_{\text{rating}}, X_{\text{age}}\ Sailors(X_{\text{sid}}, X_{\text{sname}}, X_{\text{rating}}, X_{\text{age}}) \wedge X_{\text{rating}} > 7 \wedge \exists\, X_{\text{bid}}, X_{\text{day}}\ Reserves(X_{\text{sid}}, X_{\text{bid}}, X_{\text{day}}) \wedge X_{\text{bid}} = 103\ \}$$

SQL:

3. Final projection

```
SELECT  sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid and s.rating>7 and R.bid = '103';
```

1. Implicit cross product

2. Selection on the results of cross product

Unsafe Queries, Expressive Power

- *Unsafe* queries in calculus: some queries can have an infinite number of answers.
  - e.g., $\left\{ S \mid \neg \left(S \in Sailors \right) \right\}$

- *Equivalence between RA and Safe RC*: every query that can be expressed in *relational algebra* can be expressed as *a safe query in relational calculus*; the converse is also true.

- SQL can express every query that is expressible in relational algebra/calculus.

Query Language Classes

	Algebra	Logic	SQL
Recursive Queries			
First Order Queries	RA	(safe) RC	SFW + UNION, EXCEPT no aggregation
Conjunctive Queries			

Query Language Classes

	Algebra	Logic	SQL
Recursive Queries			
First Order Queries	RA	(safe) RC	SFW + UNION, EXCEPT no aggregation
Conjunctive Queries	RA: $\sigma,\pi,\times$	Single datalog rule	S^dFW no aggregation

Conjunctive Queries (CQ)

- A subset of FO queries (i.e., less expressive).
- CQs have "better" theoretical properties than arbitrary queries.
- Query optimizer handles CQs the best--it tries to
  - flatten a nested query to a single CQ
  - break a large query into many CQs

CQ in Rule-based (Datalog) Notation

Variables

implicit ∃

$$\underbrace{\mathbf{P}(x,z)}_{\textbf{Head}} \quad \underset{\text{IF}}{\leftarrow} \quad \underbrace{\mathbf{R_1}(x, y)\ \&\ \mathbf{R_2}(y, z)}_{\textbf{Body}}$$

Subgoals **Conjunction**

- R: Extensional database (EDB) -- stored
- P: Intentional database (IDB) -- computed

Find sailors rated > 7 who have reserved boat #103

$$P(x,z) \leftarrow R_1(x, y) \,\&\, R_2(y, z)$$

$$P(X_{sname}) \leftarrow \text{Sailors}(X_{sid}, X_{sname}, X_{rating}, X_{age}) \,\&\, X_{rating}>7$$
$$\&\, \text{Reserves}(X_{sid}, 103, X_{day})$$

Properties of CQs

- **Satisfiability**
  - A query q is *satisfiable* if there exists at least one database instance D such that q(D) is non-empty.
  - Theorem: *Every CQ is satisfiable*.
- **Monotonicity**
  - A query Q is *monotonic* if for two database instances D1 and D2, D1 ⊆ D2 implies Q(D1) ⊆ Q(D2).
  - Theorem: *Every CQ is monotonic*.

Beyond First-Order Queries

	Algebra	Logic	SQL
Recursive Queries		?	
First Order Queries	RA	(safe) RC	SFW + UNION, EXCEPT no aggregation
Conjunctive Queries	RA: $\sigma,\pi,\times$	Single datalog rule	S^{d}FW no aggregation

Limitation of FO Queries

- Let D = {E(x,y)} represent a graph
- Query path(x,z) =
  - all x,z such that there is a path from x to z.

- **Theorem**: path (x,z) *cannot* be expressed in FO.

Find all of Mary's ancestors

ParentOf

Parent	Child
Mike	Joe
Joe	Alice
Joe	Bob
Alice	Mary
…	…

- Can you write a query in SQL?

SQL with Recursion

Relation to be computed recursively

```
WITH RECURSIVE Ancestor(anc, desc) AS
  ((SELECT parent AS anc, child AS desc
    FROM   ParentOf)
    UNION
  ((SELECT A.anc, p.child AS desc
    FROM   Ancestor A, ParentOf P
    WHERE A.desc = P.parent)

SELECT anc
FROM   Ancestor
WHERE desc = 'Mary';
```

Base case

Recursion

Recursive Computation

Ancestor

	Anc	Desc
Base case:	Mike	Joe
	Joe	Alice
	Joe	Bob
	Alice	Mary
Iter 1:	Mike	Alice
	Mike	Bob
	Joe	Mary
Iter 2:	Mike	Mary
Iter 3:	{ }	

ParentOf

Parent	Child
Mike	Joe
Joe	Alice
Joe	Bob
Alice	Mary

Query result:

Alice	Mary
Joe	Mary
Mike	Mary

Recursion in Datalog

Ancestor(x,y) $\leftarrow$ ParentOf(x,y)

Ancestor(x,z) $\leftarrow$ Ancestor(x,y) & ParentOf(y,z)

Use of IDB in Body

Implicit UNION

AncestorOfMary(x) $\leftarrow$ Ancestor(x, 'Mary')

A More Complete Picture

	Algebra	Logic	SQL
Recursive Queries	Fixed point operator	Datalog (recursion)	Full SQL (recursion)
First Order Queries	RA	(safe) RC	SFW + UNION, EXCEPT no aggregation
Conjunctive Queries	RA: $\sigma,\pi,\times$	Single Datalog rule	S^dFW no aggregation

Questions