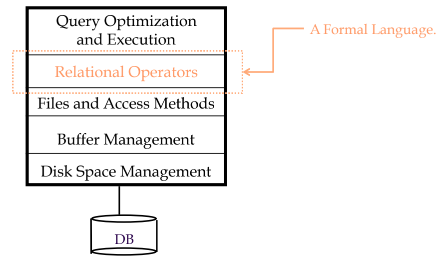
Relational Algebra

R & G, Chapter 4.2

Context



Relational Query Languages

- Query languages:
  - manipulation and retrieval of data
  - (what else is there?!)
- Relational QLs:
  - Strong formal foundation based on logic
  - Simple, powerful
  - Allow for much optimization

Why Bother with Formalism?

- We already have “physical” dataflow
  - i.e. Iterators
  - e.g. Map and Reduce
  - What more could we want?!
- *Semantic transparency*
  - With a small *domain-specific language* (DSL) for data
  - Enables rich program analysis
- As we’ll see, helps us with optimization
- Also with many other topics we won’t cover
  - Data lineage
  - Materialized views
  - Updatable views
  - ...

Relational Query Languages

Standard viewpoint: QLs != PLs
- Domain-Specific Languages for data processing
- Not Turing complete
- Not intended for complex calculations

Reality in recent years:
- Everything interesting involves a large data set
- QLs (with extensions) are quite powerful
  - A good choice for expressing *algorithms at scale*
  - An attractive choice for thinking about asynchronous and parallel programming

e.g. rx.codeplex.com, bloom-lang.org

Formal Relational QL’s

Relational Algebra:
- Operational
- Useful for representing execution plan semantics

Relational Calculus:
- A *Declarative* language (Logic!)
- Describe **what** you want, rather than **how** to compute it.
- Foundation for SQL

Preliminaries

- A query is applied to relation *instances*
- Result is also a relation instance
  - Schemas of input relations are fixed
  - Schema for query results are also fixed
    - determined by query language syntax
    - contrast with MapReduce
- Pure relational algebra has *set* semantics
  - No duplicate tuples in a relation
  - Vs. SQL, which has *multiset* semantics

Relational Algebra: 5 Basic Operations

- Selection (σ) Selects a subset of rows (horizontal)
- Projection (π) Retains only desired columns (vertical)
- Cross-product ($\times$) Allows us to combine two relations.
- Set-difference ($-$) Tuples in r1, but not in r2.
- Union ($\cup$) Tuples in r1 or in r2.

Since each operation returns a relation, operations can be composed! (Algebra is "closed")

Example Instances

R1

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

Boats

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

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

Projection (π)

- Examples: π_{age}(S2) ; $\pi_{sname,rating}$(S2)
- Retains only attributes in the "projection list"
- Schema of result:
  - the fields in the projection list
  - with the same names that they had in the input relation
- Projection operator has to eliminate duplicates
  - Note: real systems typically don't do duplicate elimination
  - Unless the user explicitly asks for it
  - (Why not?)

Projection (π)

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

S2

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

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

age
35.0
55.5

$\pi_{age}(S2)$

Selection (σ)

- Selects rows that satisfy *selection condition*.
- Result is a relation with same schema
- Do we need to do duplicate elimination?

sid	sname	rating	age
28	yuppy	9	35.0
~~31~~	~~lubber~~	~~8~~	~~55.5~~
~~44~~	~~guppy~~	~~5~~	~~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 and Set-Difference

- Two input relations, must be *union-compatible*:
  - Same number of fields.
  - "Corresponding" fields have same type.
- Duplicate elimination required?

Union

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

$S1 \cup S2$

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

Set Difference

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

$S1 - S2$

sid	sname	rating	age
22	dustin	7	45.0

$S2 - S1$

sid	sname	rating	age
28	yuppy	9	35.0
44	guppy	5	35.0

A Note on Set Difference

Most relational algebra operators are *monotonic*
- Monotonic: As input instances grow, output grows
- I.e. Consider a monotonic query Q(R1, S1, T1, …) over relation instances
- If R2 ⊃ R1, then Q(R2, S1, T1, …) ⊇ Q(R1, S1, T1, …)

Set Difference is *non-monotonic*
- Example query: S1 – R1
- "Grow" R: i.e. choose R2 ⊃ R1
- If R2 ⊃ R1, then S1 – R2 ⊆ S1 – R1

One implication: set difference => *blocking* iterator
- For S – R, need to have full contents of R before emitting any results
- *All other operators can be implemented in a non-blocking fashion!*

Cross-Product

- S1 × R1: Each row of S1 paired with each row of R1.
- Q: How many rows in the result?
- Result schema: one field per field of S1 and R1,
  - Field names "inherited" when possible.
  - Naming conflict? S1 and R1 have a field with same name.
  - Can use a *renaming* operator ρ:

$$\rho\,(C(1 \rightarrow sid1, 5 \rightarrow sid2),\ S1 \times R1)$$

Output relation name

Renaming list: position->name

Cross Product Example

R1

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

S1

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

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

Compound Operator: ∩

- In addition to 5 basic operators…
- Several "Compound Operators"
  - Add no computational power to the language
  - Useful shorthand
  - Can be expressed solely with the basic ops
- Intersection takes two input relations, which must be union-compatible
- Q: How to express it using basic operators?

$R \cap S = ?$

Compound Operator: ∩

- In addition to 5 basic operators…
- Several "Compound Operators"
  - Add no computational power to the language
  - Useful shorthand
  - Can be expressed solely with the basic ops.
- Intersection takes two input relations, which must be union-compatible.
- Q: How to express it using basic operators?

$R \cap S = R - \ldots$

Compound Operator: ∩

- In addition to 5 basic operators…
- Several "Compound Operators"
  - Add no computational power to the language
  - Useful shorthand
  - Can be expressed solely with the basic ops.
- Intersection takes two input relations, which must be union-compatible.
- Q: How to express it using basic operators?

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

Intersection

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

S1

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

S2

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

$S1 \cap S2$

Compound Operator: Join

- Involves cross product & selection
  - And sometimes projection (for natural join)
- Most common type of join: "natural join"
  - $R \bowtie S$ conceptually is:
    - Compute $R \times S$
    - *Select* rows where attributes appearing in *both* relations have equal values
    - *Project* onto all unique attributes and *one copy* of each of the common ones.
- Note: obviously we should use a good join algorithm, not a cross-product!!

Natural Join Example

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

R1

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

S1

$S1 \bowtie R1$ =

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

Other Types of Joins

Condition Join (or "theta-join"):

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

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

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

- *Result schema* same as that of cross-product
- May have fewer tuples than cross-product
- <u>*Equi-Join*</u>: Special case: condition *c* contains only conjunction of *equalities*

Examples

Reserves

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

Sailors

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

Boats

<u>bid</u>	bname	color
101	Interlake	Blue
102	Interlake	Red
103	Clipper	Green
104	Marine	Red

Find names of sailors who've reserved boat #103

- Solution 1: $\pi_{sname}((\sigma_{bid=103} Reserves) \bowtie Sailors)$
- Solution 2: $\pi_{sname}(\sigma_{bid=103}(Reserves \bowtie Sailors))$

Find names of sailors who've reserved a red boat

Information about boat color only available in Boats; so need an extra join:

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

A more efficient solution:

$$\pi_{sname}(\pi_{sid}((\pi_{bid} \sigma_{color='red'} Boats) \bowtie Res) \bowtie Sailors)$$

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

Can identify all red or green boats, then find sailors who've reserved one of these boats:

$$\rho\ (Tempboats, (\sigma_{color='red' \vee color='green'} Boats))$$

$$\pi_{sname}(Tempboats \bowtie Reserves \bowtie Sailors)$$

Find sailors who've reserved a red <u>and</u> a green boat

- Cut-and-paste previous slide?

$$\rho\ (Tempboats, (\sigma_{color='red' \wedge color='green'} Boats))$$

$$\pi_{sname}(Tempboats \bowtie Reserves \bowtie Sailors)$$

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

- Previous approach won't work!
- Must identify sailors who've reserved red boats, sailors who've reserved green boats, then find the intersection (note that *sid* is a key for Sailors):

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

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

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

Summary

- Relational Algebra: a small set of operators mapping relations to relations
  - Operational, in the sense that you specify the explicit order of operations
  - A closed set of operators! Mix and match.
- Basic ops include: σ, π, $\times$, $\cup$, $-$
- Important compound ops: $\cap$, $\bowtie$