

ICS 321 Fall 2011

The Relational Model of Data (i)

Asst. Prof. Lipyeow Lim

Information & Computer Science Department

University of Hawaii at Manoa

Data Models

A data model is a collection of concepts for describing data

- Structure of the data.
 - More of a *conceptual model* rather than a *physical data model*. Eg. Arrays, objects in C/C++
- Operations on the data
 - *Queries* and *modifications* only
- Constraints on the data
 - Limitations on the data. Eg. Data type etc.

Examples: the relational model and the semi-structured model (XML)

The Relational Model

- *Relational database*: a set of *relations*
- A *relation* is made up of 2 parts:
 - *Instance* : a *table*, with rows and columns.
#Rows = *cardinality*, #fields = *degree / arity*.
 - *Schema* : specifies name of relation, plus name and *domain/type* of each column or attribute.
 - E.G. Students(sid: string, name: string, login: string, age: integer, gpa: real).
- Can think of a relation as a *set* of rows or *tuples* (i.e., all rows are distinct).

Example Instance of Students Relation

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

- Cardinality = 3, degree=5, all rows distinct
- Do all columns in a relation instance have to be distinct?

Relational Query Languages

- A major strength of the relational model
 - Supports simple, powerful *querying* of data.
- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
 - The key: precise semantics for relational queries.
 - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.
- Query Languages **!=** programming languages!
 - QLs not expected to be “Turing complete”.
 - QLs not intended to be used for complex calculations.
 - QLs support easy, efficient access to large data sets.

Formal Relational Query Languages

- Two mathematical Query Languages form the basis for “real” languages (e.g. SQL), and for implementation:
 - Relational Algebra: More **operational**, very useful for representing execution plans.
 - Relational Calculus: Lets users describe what they want, rather than how to compute it. (**Non-operational, more declarative.**)

Preliminaries

- A query is applied to *relation instances*, and the result of a query is also a relation instance.
 - *Schemas of input* relations for a query are **fixed** (but query will run regardless of instance!)
 - The **schema for the result** of a given query is also **fixed!** Determined by definition of query language constructs.
- Positional vs. named-field notation:
 - Positional notation easier for formal definitions, named-field notation more readable.
 - Both used in SQL

Example Relational Instances

- “Sailors” and “Reserves” relations for our examples.
- We’ll use positional or named field notation, assume that names of fields in query results are ‘inherited’ from names of fields in query input relations

R1

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

S1

<u>sid</u>	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

S2

<u>sid</u>	sname	rating	age
28	Yuppy	9	35.0
31	Lubber	8	55.5
44	Guppy	5	35.0
58	Rusty	10	35.0

Relational Algebra

- Basic operations:
 - Selection (σ) Selects a subset of rows from relation.
 - Projection (π) Deletes unwanted columns from 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:
 - Intersection, join, division, renaming: Not essential, but (very!) useful.
- Since each operation returns a relation, **operations can be composed!** (Algebra is “closed”.)

Projection

- Deletes attributes that are not in *projection list*.
- **Schema** of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate *duplicates*! (Why??)
- Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)

$\Pi_{\text{sname, rating}} (\mathbf{S2})$

sname	rating
Yuppy	9
Lubber	8
Guppy	5
Rusty	10

$\Pi_{\text{age}} (\mathbf{S2})$

age
35.0
55.5
35.0
35.0

Selection

- Selects rows that satisfy *selection condition*.
- No duplicates in result! (Why?)
- *Schema* of result identical to schema of (only) input relation.
- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition*.)

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

<u>sid</u>	sname	rating	age
28	Yuppy	9	35.0
31	Lubber	8	55.5
44	Guppy	5	35.0
58	Rusty	10	35.0

$\Pi_{\text{sname, rating}} (\sigma_{\text{rating} > 8} (\mathbf{S2}))$

<u>sid</u>	sname	rating	age
28	Yuppy	9	35.0
31	Lubber	8	55.5
44	Guppy	5	35.0
58	Rusty	10	35.0

Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be **union-compatible**:
 - Same number of fields.
 - ‘Corresponding’ fields have the same type.
- What is the **schema** of result?

S1

<u>sid</u>	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

S1 U S2

<u>sid</u>	sname	rating	age
22	Dustin	7	45.0
28	Yuppy	9	35.0
31	Lubber	8	55.5
44	Guppy	5	35.0
58	Rusty	10	35.0

S2

<u>sid</u>	sname	rating	age
28	Yuppy	9	35.0
31	Lubber	8	55.5
44	Guppy	5	35.0
58	Rusty	10	35.0

Intersection & Set-Difference

$S1 \cap S2$

<u>sid</u>	sname	rating	age
31	Lubber	8	55.5
58	Rusty	10	35.0

$S1 - S2$

<u>sid</u>	sname	rating	age
22	Dustin	7	45.0

S1

<u>sid</u>	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

S2

<u>sid</u>	sname	rating	age
28	Yuppy	9	35.0
31	Lubber	8	55.5
44	Guppy	5	35.0
58	Rusty	10	35.0

Cross-Product

- Consider the cross product of S1 with R1
- Each row of S1 is paired with each row of R1.
- *Result schema* has one field per field of S1 and R1, with field names `inherited` if possible.
 - *Conflict*: Both S1 and R1 have a field called *sid*.
 - Rename to *sid1* and *sid2*

R1	<u>sid</u>	<u>bid</u>	<u>day</u>
	22	101	10/10/96
	58	103	11/12/96

S1	<u>sid</u>	sname	rating	age
	22	Dustin	7	45.0
	31	Lubber	8	55.5
	58	Rusty	10	35.0

S1 × R1

sid	sname	rating	age	sid	bid	day
22	Dustin	7	45	22	101	10/10/96
22	Dustin	7	45	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

Renaming

- The expression:

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

- Renames the result of the cross product of S1 and R1 to “C”
- Renames column 1 to sid1 and column 5 to sid2

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

sid1	sname	rating	age	sid2	bid	day
22	Dustin	7	45	22	101	10/10/96
22	Dustin	7	45	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

Joins

- Condition Join: $R \bowtie_c S = \sigma_c(R \times S)$
- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently
- Sometimes called a *theta-join*.

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

sid	sname	rating	age	sid	bid	day
22	Dustin	7	45	58	103	11/12/96
31	Lubber	8	55.5	58	103	11/12/96

Equi-Joins & Natural Joins

- **Equi-join**: A special case of condition join where the condition c contains only *equalities*.
 - **Result schema** similar to cross-product, but only one copy of fields for which equality is specified.
- **Natural Join**: Equi-join on *all* common fields.

$$S1 \bowtie_{sid} R1$$

sid	sname	rating	age	bid	day
22	Dustin	7	45	101	10/10/96
58	Rusty	10	35.0	103	11/12/96

Division

- Not supported as a primitive operator, but useful for expressing queries like:
 - Find sailors who have reserved all boats.*
- Let A have 2 fields, x and y ; B have only field y :
 - $A/B = \{ \langle x \rangle \mid \exists \langle x, y \rangle \in A \ \forall \langle y \rangle \in B \}$
 - i.e., **A/B contains all x tuples (sailors) such that for every y tuple (boat) in B , there is an xy tuple in A .**
 - *Or:* If the set of y values (boats) associated with an x value (sailor) in A contains all y values in B , the x value is in A/B .
- In general, x and y can be any lists of fields; y is the list of fields in B , and $x \cup y$ is the list of fields of A .

Examples of Division

P

Col1	Col2
A	1
A	2
A	3
A	4
B	1
B	2
C	2
D	2
D	4

Q

Col2
2

R

Col2
2
4

S

Col2
1
2
4

P / Q

Col1
A
B
C
D

P / R

Col1
A
D

P / S

Col1
A

Expressing A/B Using Basic Operators

- Division is not essential op; just a useful shorthand.
 - (Also true of joins, but joins are so common that systems implement joins specially.)
- *Idea*: 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 .
 - Disqualified x values : $\pi_x ((\pi_x (A) \times B) - A)$
 - A/B : $\pi_x (A) -$ all disqualified tuples

Find names of sailors who've reserved boat #103

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

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

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

$\pi_{sname}(Temp2)$

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

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:

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

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

- Can also define Tempboats using union! (How?)
- What happens if \vee is replaced by \wedge in this query?

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*):

$$\rho (Tempred, \pi_{sid}((\sigma_{color='red'} Boats) \bowtie Reserves))$$
$$\rho (Tempgreen, \pi_{sid}((\sigma_{color='green'} Boats) \bowtie Reserves))$$
$$\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$$

Find the names of sailors who've reserved all boats

- Use division; schemas of the input relations to / must be carefully chosen:

$$\rho (Tempsids, (\pi_{sid, bid} Reserves) / (\pi_{bid} Boats))$$

$$\pi_{sname} (Tempsids \bowtie Sailors)$$

Find sailors who've reserved all 'Interlake' boats

- Same as previous, but put a selection on Boats:

... / $\pi_{bid}(\sigma_{bname='Interlake'} Boats)$

Summary

- The Relational Data Model
- Two theoretical relational query languages: relational algebra & relational calculus
- Relational Algebra (RA) operators: selection, projection, cross-product, set difference, union, intersection, join, division, renaming
- Operators are closed and can be composed
- RA is more operational and could be used as internal representation for query evaluation plans.
- For the same query, the RA expression is not unique.
- Query optimizer can choose the most efficient version.