Relational Query Languages I

Relational Algebra

What is an “Algebra”

- Mathematical system
  - *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?

- An algebra whose operands are relations or variables that represent relations
- Operators are designed to do the most common things that we need to do with relations in a database
  - The result is an algebra that can be used as a *query language* for relations

Relational Query Languages (1)

- 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
  - precise semantics for relational queries
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.

Relational Query Languages (2)

- *<u>Query languages</u>:* Allow manipulation and retrieval of data from a database
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic
  - Allows for optimization
- Query Languages != programming languages!
  - 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, *declarative*.)

The SQL Query Language (1)

- Developed by IBM's System-R Project in the 1970s
- Need for a standard since it is used by many vendors
- Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision)
  - SQL-99 (major extensions, current standard)

The SQL Query Language (2)

- SQL has been influenced by both Relational Algebra (RA) & Relational Calculus (RC)
- The other variant of RC is Domain Relational Calculus (DRC) which has greatly influenced Query By Example (QBE)

The SQL Query Language (3)

- SQL consists of:
  - DDL (Data Definition Language)
    - Create conceptual schema
  - DML (Data Manipulation Language)
    - Relational operators
    - Insert, Delete, Update
  - VDL (View Definition Language)
    - Specify user views & their mapping to the conceptual schema (in most DBMSs, done by DDL)
  - SDL (Storage Definition Language)
    - File organization
    - Indexes

Operations on Relations

- Restrict/Select
- Project
- Join

Relational Operations

- Divide

- Union
- Intersection
- Difference
- Product

Set Operations

Relational Algebra Overview (Elmasri & Navathe)

Relational Algebra consists of several groups of operations

- Unary Relational Operations
  - SELECT (symbol: σ (sigma))
  - PROJECT (symbol: π (pi))
  - RENAME (symbol: ρ (rho))
- Relational Algebra Operations From Set Theory
  - UNION ($\cup$), INTERSECTION ($\cap$), DIFFERENCE (or MINUS, –)
  - CARTESIAN PRODUCT (**x**)
- Binary Relational Operations
  - JOIN (several variations of JOIN exist)
  - DIVISION
- Additional Relational Operations
  - OUTER JOINS, OUTER UNION
  - AGGREGATE FUNCTIONS (These compute summary of information: for example, SUM, COUNT, AVG, MIN, MAX)

Relational Algebra

- Collection of operations on relations
- 8 operators + Rename operator
- Codd’s 8 operators do not form a minimal set
- Some of them are not primitive
- Join, Intersect, & Divide can be defined in terms of other 5
- None of these 5 can be defined in terms of the remaining 4
- MINIMAL SET
- Join, Intersect, & Divide are important & should be supported directly

Relational Algebra

- Some common DB requests can not be performed with the RA operations
- Additional operations are developed
- Aggregate functions & additional types of joins

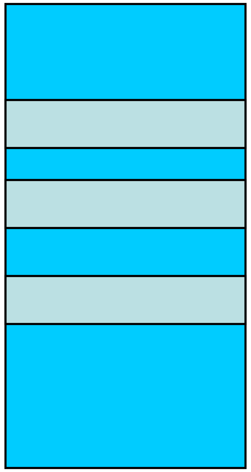
Relational Algebra

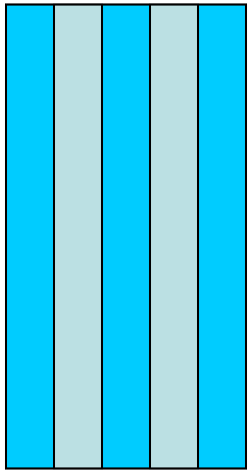
- Provides a formal foundation for relational model operations
- Plays a pivotal role in query optimization in RDBMSs
- Some of its concepts are implemented in SQL
- Expressive power of RA is used as a metric of how powerful a relational DB query language is

Relational Completeness

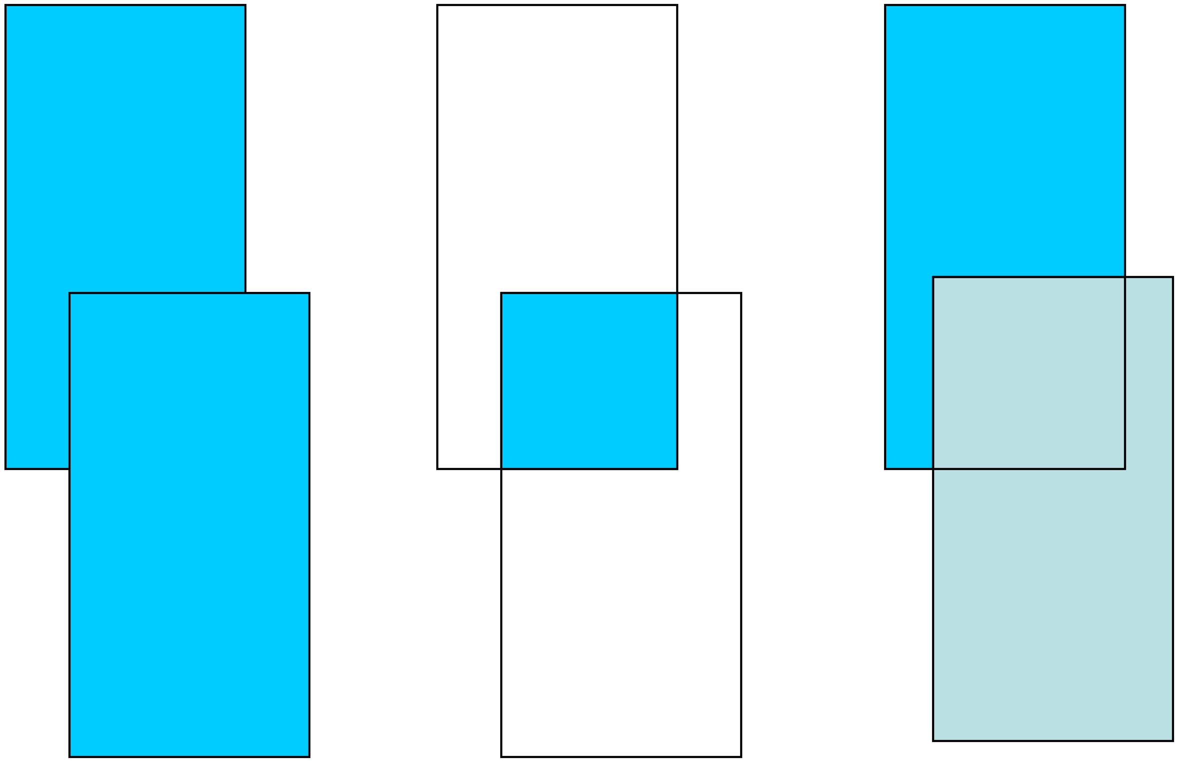
- A relational DB query language is said to be Relationally Complete, if it can express all queries that we can express in RA
- Relational DB query languages are expected to be relationally complete
- Commercial query languages like SQL, support features that allow us to express some queries that are not possible in RA

Restrict & Project





Union, Intersection & Difference(1)



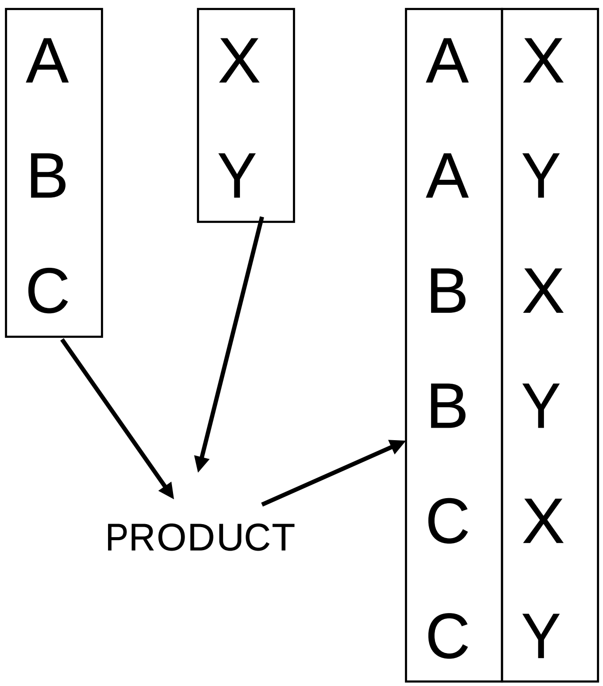
Union, Intersection & Difference(2)

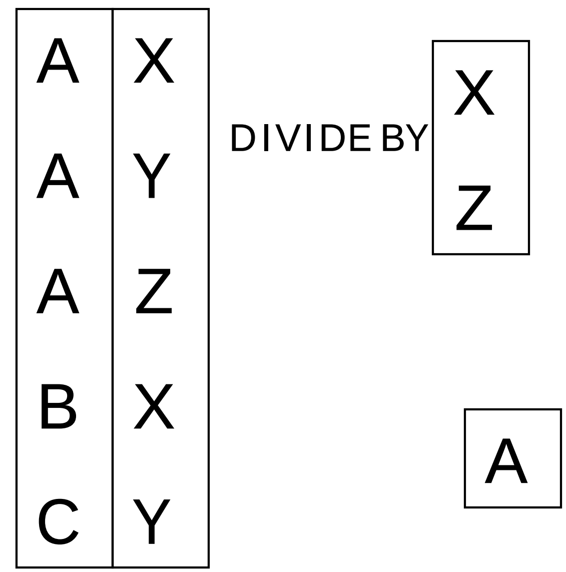
Union Compatibility: r U s is valid if:

- Relations r & s have the same arity
- Domains of the i^{th} attribute of r is the same as the domain of the i^{th} attribute of s, $\forall\, i$.

  Note that r & s can be either database relations or derived relations

Product & Divide





Divide

DIVIDE

S1	P1
S1	P2
S2	P3
S2	P4
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4

Shipment

P2

P2
P4

P1
P2
P4

Parts

S1
S2
S3
S4

S2
S4

S2

Join

A1	B1
A2	B1
A3	B2

B1	C1
B2	C2
B3	C3

NATURAL JOIN

A1	B1	C1
A2	B1	C1
A3	B2	C2

Closure

- A relation is closed under relational and set operators
- The result of these operators on relation(s) is another relation
- Output from one operation can become input to another
- Nested Expressions possible

Case Study: Sailors Database

- Sailors (<u>sid:integer</u>, sname:string,rating:integer,age:real)
- Boats (<u>bid:integer</u>,bname:string,color:string)
- Reserves (<u>sid:integer, bid:integer,day:date</u>)

Example Schema

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

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

- 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.

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

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

age
35.0
55.5

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

Selection

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

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

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

sname	rating
yuppy	9
rusty	10

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

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 domain/data type

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

$S1 \cup S2$

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

$S1 \cap S2$

Union, Intersection & Difference

Union Compatibility: r U s is valid if:

- Relations r & s have the same arity
- Domains of the i^{th} attribute of r is the same as the domain of the i^{th} attribute of s, $\forall\, i$.

  Note that r & s can be either database relations or derived relations

Cross-Product

- 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*.

(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

- *Renaming operator*: $\rho(C(1\rightarrow sid1, 5\rightarrow sid2), S1\times R1)$

Joins

- ***Condition 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.
- Fewer tuples than cross-product, might be able to compute more efficiently
- Sometimes called a *theta-join*.

Joins

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

Examples of Division A/B

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

A

pno
p2

B1

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

sno
s1
s4

A/B2

sno
s1

A/B3

Extended Operators of RA

- Duplicate-Elimination Operator δ
- Aggregation Operators
    - Sum
    - Count
    - Average
    - Max
    - Min
- Grouping Operator
- Sorting Operator τ
- Extended Projection
- Outerjoin Operator

Aggregate Functions & Operations

- Aggregation function takes a collection of values and returns a single value as a result.

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- Aggregate operation in relational algebra: General Form

$$\boxed{{}_{G_1, G_2, \ldots, G_n}\, \vartheta_{F_1(A_1), F_2(A_2, \ldots, F_n(A_n)}(E)}$$

  E is any relation
  - G_1, G_2 …, G_n is a list of attributes on which to group (optional)
  - Each F_i is an aggregate function
  - Each A_i is an attribute name

Aggregate Operation – Example

Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

$\mathcal{G}_{\mathbf{sum(c)}}(\mathrm{r})$

sum(*C*)
27

Aggregate Operation – Example

Relation *account* grouped by *branch-name*:

branch_name	*account_number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$_{\textit{branch\_name}}\,\mathcal{G}_{\,\textbf{sum}(\textit{balance})}\,(\textit{account})$

branch_name	**sum**(*balance*)
Perryridge	1300
Brighton	1500
Redwood	700

Aggregate Functions (Cont.)

- Result of aggregation does not have a name
  - Can use rename operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation

$$_{\textit{branch\_name}}\, \mathcal{G}\, _{\textbf{sum}(\textit{balance})\ \textbf{as}\ \textit{sum\_balance}}(\textit{account})$$

Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the projection list.

$$\prod_{F_1, F_2, \ldots, F_n}(E)$$

- E is any relational-algebra expression
- Each of F_1, F_2, …, F_n are are arithmetic expressions involving constants and attributes in the schema of E.
- Given relation *credit_info(customer_name, limit, credit_balance),* find how much more each person can spend:

$$\prod_{customer\_name,\ limit\ -\ credit\_balance} (credit\_info)$$

Generalized Projection

Cust_name	*limit*	*Credit_bal*
Curry	2000	1750
Hayes	1500	1500
Jones	6000	700
Smith	2000	400

Given relation *credit_info(customer_name, limit, credit_balance),* find how much more each person can spend:

$$\Pi_{customer\_name,\ (limit - credit\_balance)\ as\ credit\_available}\ (credit\_info)$$

Cust_name	*Credit_available*
Curry	250
Hayes	0
Jones	5300
Smith	1600

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist

Outer Join – Example

Relation *loan*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Relation *borrower*

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

Outer Join – Example

Inner Join

loan $\bowtie$ *Borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

Left Outer Join

loan ⟕ ***Borrower***

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

Outer Join – Example

Right Outer Join

loan ⟖ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

Full Outer Join

loan ⟗ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist or something that is NA
- Aggregate functions simply ignore null values (as in SQL)
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same (as in SQL)