Relational Algebra

σ ← select
Π ← project
∪ ← union
A − B ← set diff.
A × B ← product
ρ ← rename

Summary

- $E_1 \cup E_2$
- $E_1 - E_2$
- $E_1 \times E_2$
- $\sigma_p(E_1)$, P is a predicate on attributes in E_1
- $\Pi_L(E_1)$, L is a list consisting of some of the attributes in E_1
- $\rho_x(E_1)$, x is the new name for the result of E_1

Select

Notation: $\sigma_p(r)$ — p: select requirements; r: Table

A op B or C
op: =, ≥, ≠, < etc.

ex. "σ dept_name = "Physics" (Instructor)"

• commutative & associative.

Project

Notation: $\Pi_{A_1, A_2 \ldots A_K}(r)$ — attribute names

every column that is not listed within attributes are removed. ... if you only want to output X, Y, Z list them as attributes

Since it is a set duplicates are removed

Relation r:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

$\pi_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Union

Notation: r ∪ s (Removes Dups)

- r, s have to have same # of columns
- column type must be compatable

ex:

$\Pi_{customer-name}$ (depositor) ∪ $\Pi_{customer-name}$ (borrower)

Same type + same # of columns

Returns type column

Set Difference

Notation: r − s

- r and s must have same # of columns
- types must be compatable

r s

r − s

Relations r, s:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

r − s

A	B
α	1
β	1

ex: $\Pi_{customer-name}$ (depositor) − $\Pi_{customer-name}$ (Borrower)

match # & type

Returns type column

Cartesian-Product

notation: r × s

- if r, s disjoint; r ∩ s = ∅ → no renaming
  else → renaming

every possible combination

Relations r, s:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

r x s:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

A	B	C	D	E
a	1	a	10	a
b	2	a	10	a
a	1	B	10	a
b	2	B	10	a
a	1	B	20	b
B	1	B	20	b
a	1	γ	10	b
B	1	γ	10	b

Rename

notation: $\rho_x(E)$ returns result of E under name X

if E has N columns:

$\rho_{x(A_1, A_2 \ldots A_n)}(E)$

returns w attributes renamed to $A_1, A_2 \ldots A_n$

Overpowered can be used to rename result + columns

Extras

Functions: avg, min, max, sum, count

Relation r:

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

$g_{\text{sum(c)}}(r)$

Sum(C)
27

Aggregation

- $_{G1, G2, \ldots, Gn}\, g_{F1(A1), F2(A2), \ldots, Fn(An)}(E)$

E is an expression in RA

$G_1, G_2 \ldots, G_n$ is a list of attributes for grouping

Every F_i is an aggregation function

Every A_i is the name of an attribute

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

branch-name $g_{sum(balance)}$ (*account*)

branch-name	sum(balance)
Perryridge	1300
Brighton	1500
Redwood	700

Additional Operations

A ∩ B ← set intersection
⋈ ← natural join
A/B ← Division
← ← Assignment

Set-Intersection

notation: r ∩ s

Same laws as union

Joey-Issu-2024

Relation r, s:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

r ∩ s

A	B
α	2

ex: $\Pi_{cus-name}$ (deposit) ∩ $\Pi_{cus-name}$ (borrower)

Natural join

r ⋈ s

Ex: $\Pi_{cus-name, branch-name}$ (deposit ⋈ account)

- result removes duplicates (if both tables have them)
- extending table (combining them)

if no match between tables it gets ignored (only include it if it matches row from another table)

Relations r, s:

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

s

r ⋈ s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

* Key is: column in R & S must have same name.

Division

Notation: R/S

ex: R = ($A_1 \ldots A_K$, $B_1 \ldots B_K$)
S = ($B_1 \ldots B_K$)

R/S = $A_1 \ldots A_K$ where there is a value that corresponds to each value in column S.

* useful for "for all" queries

Relations r, s:

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
ε	6
ε	1
β	2

r

B
1
2

s

r / s:

A
α
β

Assignment

Notation: ←

Allows for complex queries as it stores value in temp variable

temp1 ← $\Pi_{R-S}(r)$

Types of Join

- join (Θ)
- equality join
- Semi-join
- outer join

Θ-join

Notation: r ⋈$_c$ s

A Θ-join is equivalent to a cartesian product that is followed by a selection with a condition **c**

$R \bowtie_c S = \sigma_c(R \times S)$

returns all Attributes of R & S

Basically puts condition on Natural join

EMPLOYEE(eid, ename, esalary, mngrId)
MANAGER(mid, mname, msalary)
Find the names of the employees with a salary bigger than the salary of their managers

Π_{ename}(EMPLOYEE ⋈$_{mngrId = mid \text{ and } esalary > msalary}$ MANAGER)

Equality join

Notation: r ⋈$_c$ s

Combine tuples that match (same value) on some attributes

Ex:
EMPLOYEE ⋈$_{mngrid=mid}$ Manager

Semi-join

Notation: r ⋉$_c$ s

Select a subset of a join; think join + projection

Example: Find all the information about employees that are managers in some department

EMPLOYEE ⋉$_{SSN=MgrSSN}$ DEPARTMENT

Outer join

extension of join to not Lose extra info.
if No match → then null to missing value

Outer join is distinguished as:
Left outer join: R ⟕ S (all tuples of R are present in the result)
Right outer join: R ⟖ S (all tuples of S are present in the result)
Full outer join: R ⟗ S (all tuples of R and S are present in the result)

loan

loan-number	branch-name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

borrower

customer-name	loan-number
Jones	L-170
Smith	L-230
Hayes	L-155

Inner Join
loan ⋈ 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

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

Definition of a view

A view is defined with the command **create view** :

create view *v* **as** <query expression>

where <query expression> is the expression of a query. The name of the view is *v*.

From the moment that a view is defined, its name is used to refer ot the virtual relations that it denotes.

The definition of the view does not create tuples in the new relation.

- Oppositely, the definition of the view means that an expression is created and maintained, which, during execution, is substituted in the queries that use it.

Ex: **create view** *all-customer* **as** — Lazy variable
$\Pi_{branch-name,\ customer-name}$ (depositor ⋈ account)
∪ $\Pi_{branch-name,\ customer-name}$ (borrower ⋈ loan)

- We can find all the customers of Perryridge branch by writing:

$\Pi_{customer-name}(\sigma_{branch-name = \text{"Perryridge"}}(all\text{-}customer))$

Tuple Relational Calculus

Review of Discrete Math
— get Good 😎

Domain Relational Calculus

Set/graph w function Applied to it:

Domain Relational Calculus

- A nonprocedural query language equivalent in power to the tuple relational calculus
- Each query is an expression of the form:

$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n) \}$

- $x_1, x_2, \ldots, x_n$ represent domain variables
- P represents a formula similar to that of the predicate calculus